TR3500 - 5

Stellingen behorend bij het proefschrift *Efficiënte Algoritmes voor het Numeriek Oplossen van Differentiaalvergelijkingen*

Jason Frank

1. Vanwege de conclusie van Hoffmann (*Computing* 41 (1989), 335–348) dat het twee keer toepassen van het klassieke Gram-Schmidt (KGS) proces leidt tot een verzameling van vektoren die orthogonaal zijn tot machinenauwkeurigheid, en vanwege de conclusie van Hoofdstuk 3 van dit proefschrift dat het rekenwerk voor het twee keer toepassen van KGS ruwweg hetzelfde is als het rekenwerk voor Householder orthogonalisatie, en vanwege het feit dat de implementatie van het Householder algoritme veel complexer is dan van KGS, moet men zich afvragen of het niet tijd is voor Householder zich terug te trekken naar de leerboeken.

2. Gegeven zij een $n \times n$ beneden driehoeksmatrix X. Defineer $J(X)$ als een n-vektor waarvan het $i$de element gegeven wordt door de kolomindex van het meest-rechtse niet-nul element in de $i$de rij van X. Dan geldt als M een nilpotente beneden driehoeksmatrix is en N ook een beneden driehoeks matrix is, dat $J(MN) \leq J(M)$ en $J(NM) \leq J(M)$, met elementsgewijze toepassing van $\leq$.

3. Zij C een symmetrische positief semidefiniete matrix, en $P = P^2$ een projectie. Dan geldt: indien PC symmetrisch is, dan is PC positief semidefiniet.

4. Aangenomen dat samenwerken het doel is van het wetenschappelijk publiceren, biedt het Internet een ideaal forum voor actuele ontwikkelingen. Het is te betreuren dat de "publish or perish" werkelijkheid van de meeste wetenschappelijke betrekkingen het gebruik van het Internet voor dit doel ontmoedigt.

5. De driehoekige vorm van het bibliotheek gebouw van de T.U. Delft is in conflict met de menselijke vorm. Blijkbaar is dit een statement tegen de ergonomische beweging in design; net zoals de treden van de trap die anderhalve stapgrootte breed zijn, de acht voordeuren waarvan vier altijd op slot zijn, het tegelpad in de omgeving waar men elke derde stap in een gat komt, en het onhandige kelderarchief waar men een half uur moet wachten op een tijdschrift van vorig jaar terwijl er boeken van vijftig jaar oud op de planken staan.

6. Er zou geen leenstatus "Permanent niet beschikbaar" hoeven te zijn voor een boek in de catalogus van de T.U. Delft bibliotheek.

7. De traditionele opzet van het telefoonnetwerk is gebaseerd op vaste verbindingen tussen gebouwen, terwijl het eigenlijk gaat om communicatie tussen mensen. In dit opzicht is de mobiele telefoon een veel natuurlijker idee. Men belt namelijk niet om met een gebouw maar meestal om met een specifiek persoon te spreken. Daarom gaat de vaste telefoon verdwijnen.

8. Door naar *Amazon.com* te kijken, krijgt men een indruk van het enorme aantal beschikbare (papieren) publicaties. Tel daarbij op de hoeveelheid papier die over het gemiddelde bureau en door de brievenbus komt, dan is de 20e eeuw zeker "de papieren eeuw" te noemen. Het is te hopen dat er in de huidige eeuw meer gebruik wordt gemaakt van elektronische media.

9. Het handmatige afdrukken van foto's zal een verloren kunst worden in de 21e eeuw. Ooit zal het mogelijk worden een ambachtsman in 20e eeuwse dracht in een museum foto's te zien ontwikkelen voor een "papieren krant".

10. Er is een subtiel verschil tussen het ten onrechte vaak als equivalent beschouwde Engelse *ready* en het Nederlandse *klaar*. *Ready* betekent eigenlijk *klaar voor*, maar nooit *klaar mee* (= *finished*). Dit kan tot een grappige fout leiden in de logica van een computer programma als de conditie van een "while loop" vertaald wordt als: `while (.not. ready) do ...`

3500
739408
3 [illegible]
TR 3500

Efficient Algorithms for the Numerical Solution of Differential Equations

Efficient Algorithms for the Numerical Solution of Differential Equations

PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus prof. ir. K. F. Wakker,
in het openbaar te verdedigen ten overstaan van een commissie
door het College voor Promoties aangewezen,

op maandag 17 april 2000 te 13.30 uur

door

Jason Edward FRANK,

Master of Science in Aerospace Engineering, University of Kansas
geboren te Hutchinson, Kansas,
de Verenigde Staten van Amerika in 1970.

Dit proefscrift is goedgekeurd door de promotoren:
Prof. dr. ir. P. Wesseling
Prof. dr. P. J. van der Houwen

Samenstelling promotiecommissie:

Rector Magnificus,	voorzitter
Prof. dr. ir. P. Wesseling,	Technische Universiteit Delft, promotor
Prof. dr. P. J. van der Houwen,	Universiteit van Amsterdam, promotor
Dr. ir. C. Vuik,	Technische Universiteit Delft, toegevoegd promotor
Prof. dr. ir. A. W. Heemink,	Technische Universiteit Delft
Prof. dr. H. A. van der Vorst,	Universiteit Utrecht
Prof. dr. A. E. P. Veldman,	Rijksuniversiteit Groningen
Dr. J. Verwer,	Centrum voor Wiskunde en Informatica

Ir. A. Segal heeft als begeleider in belangrijke mate aan de totstandkoming van het proefschrift bijgedragen.

Een deel van het promotieonderzoek werd uitgevoerd bij het CWI in het kader van het STW-project CWI-4533.

Preface

The papers comprising the chapters of this dissertation were written during my appointment at Delft University of Technology from 1996 to 1999 and during two visits to CWI in Amsterdam—one in 1995 and one from 1998 to 1999.

Chapters 2 through 6 have all been published elsewhere or have been submitted for publication as:

- J. Frank, W. Hundsdorfer and J. Verwer. On the stability of implicit-explicit linear multistep methods. *Applied Numerical Mathematics*, 25(1997):193–205.
- J. Frank and C. Vuik. Parallel implementation of a multiblock method with approximate subdomain solution. *Applied Numerical Mathematics*, 30(1999):403–423.
- C. Vuik, J. Frank, and A. Segal. A parallel implementation of the block preconditioned GCR method. To appear, *Future Generation Computer Systems.*
- J.E. Frank and P.J. van der Houwen. Parallel iteration of the extended backward differentiation formulas. To appear, *IMA Journal of Numerical Analysis.*
- J.E. Frank and P.J. van der Houwen. Diagonalizable Extended Backward Differentiation Formulas. To appear, *BIT.*

Since the chapters were written as autonomous articles, and in consideration of the breadth of the subject matter, I have decided not to attempt to define a uniform notation throughout the dissertation. Instead, I invite the reader to set the book aside and enjoy a cup of coffee between chapters so that a change in notation will not be too disturbing.

Acknowledgements

I consider myself very fortunate to have had the opportunity to work at several institutions, and with many talented colleagues since starting my PhD studies in Kansas in 1994.

Foremost I am grateful to my *promotor* Pieter Wesseling for carefully reading my reports, and encouraging outside collaborations and travel. He maintained an open mind for some drastic changes of direction in the subject of my research. Additionally, I appreciate his ability to generate interesting discussions on topics outside of work.

My *co-promotor* Pieter van der Houwen was very patient with me when I began working with his group. I learned a lot in a short time there and will miss the regular group meetings to discuss the projects of the various members. I would also like to thank the other members of the STW group: Walter Hoffmann, Walter Lioen, and Ben Sommeijer for their interest and contributions to the work of Chapters 5 and 6.

Kees Vuik has been a good collaborator, always remaining optimistic about cooperations, even when I was not, and has given me many good leads; all the while gracefully handling repeated intrusions into his office.

It is a pleasure to work with someone like Guus Segal, whose professionalism and skepticism are so well complemented by a relaxed good humor.

Besides being a stimulating co-author, Willem Hundsdorfer arranged for my first stay at CWI, based solely on one meeting and a couple of e-mails. It would have been very difficult for me to find a Ph.D. appointment without Willem's help and the support of Jan Verwer.

Ben Leimkuhler introduced me to numerical mathematics and invited me to pusue a PhD. His enthusiasm for the field continues to be inspiring, and as my first advisor, he did the most to shape my ideas of how research should be approached.

Jacques de Swart is always full of advice. In the rare case that he *doesn't* have an opinion on a subject he will take one, just to initiate a discussion. Among other things, such conversations have helped me a lot in making career-related decisions. I am grateful for his involvement in arranging the second cooperation with CWI, and for the use of his LaTeX style files.

Additionally I would mention a number of colleagues whose contributions to this dissertation are less obvious, but whose daily interaction certainly helped to shape ideas and to make the work experience humane. They are my roommates Hester Bijl, Harald van Brummelen and Duncan van der Heul, and, standing out among many others, Han Adriaens and Charles Moulinec.

The help provided by Duncan van der Heul and Kees Vuik with the translations of the summary and the *stellingen* is gratefully acknowledged.

I am *also* very grateful to my wife Linda for helping with the translations, and most especially for making the period of my Ph.D. studies an adventure.

Amsterdam, November 1999

Contents

Preface v

Contents vii

1. Introduction 1

2. On the stability of implicit-explicit linear multistep methods 11

3. A parallel multiblock method for Poisson's equation 25

4. Parallel block-preconditioned GCR for incompressible flows 51

5. Parallel extended backward differentiation formulas 67

6. Diagonalizable extended backward differentiation formulas 87

Appendix A: The parallel/multiblock structure of DeFT 107

Summary 123

Samenvatting 125

Curriculum vitae 127

Chapter 1

Introduction

In the field of parallel computing—concerned with solving computational problems by using several cooperating processors—efficiency is clearly defined. The *speedup* of a parallel numerical algorithm is defined by

$$\textit{speedup} = \frac{T_1}{T_P}.$$

where T_P is the time needed to execute the algorithm in parallel on P processors and T_1 is the time needed to execute the *best-known* serial algorithm which solves the same problem. The *parallel efficiency* is defined to be the speedup divided by the number of processors:

$$\textit{efficiency} = \frac{T_1}{T_P P}. \tag{1.1}$$

A parallel efficiency of 100% is only achieved for certain trivial (from a parallel computing point of view) examples consisting of many fully decoupled problems, that is, problems which require no real cooperation between concurrent processes.

The efficiency of a serial numerical algorithm is not as well defined. If one were to define serial efficiency by restricting the definition (1.1) to the one-processor limit case, one would define the efficiency of a serial algorithm by comparing its time to that of the *best known* serial algorithm (and if the result were less than 100%, one would promptly switch algorithms).

Obviously this is not really a very useful definition for comparing the efficiencies of serial algorithms. The dictionary [3] defines efficiency as "effective operation as measured by a comparison of production with cost (as in energy, time, and money)." The problem with applying the definition (1.1) to serial algorithms is that it assumes that the *production* of both algorithms is the same, and in that case efficiency is inversely proportional to cost—measured in seconds of computing effort.

A more sophisticated definition of the efficiency of a numerical algorithm would include a measure of the production, defined with respect to the accuracy, precision or qualitative agreement of the numerical result. The cost should ideally reflect not only the use of computing resources, but also that of the human resources (if the fastest known serial program happens to require user intervention every 10 minutes for 8 hours, it may not be considered to be a very efficient one by the personnel department). Probably, at the bottom line, the cost of a program should be measured in dollars (Euros?).

In many cases, we can leave the definition of efficiency vague, and consider only the *relative efficiencies* of two algorithms. If the products are the same but the costs differ (the assumption in the definition of parallel efficiency), or vice versa, if the costs are the same but the products differ, then a direct relative comparison is possible, as well as in the rare case that one algorithm provides *more* for *less* cost than the other (i.e. the "something for nothing" phenomenon). In general we consider the relative efficiencies of algorithms in this dissertation.

1.1 Numerical integration of differential equations

This dissertation concerns the design of efficient computational methods for approximating the solution of systems of ordinary differential equations (ODEs), for which the prototype is the initial value problem:

$$\begin{aligned} &y'(t) = f(t, y(t)), \\ &t \in \mathbb{R}, \;\; y : \mathbb{R} \to \mathbb{R}^d, \\ &y(0) = y_0, \end{aligned} \tag{1.2}$$

with the derivative function f continuous and satisfying a Lipschitz condition. Problems of this form are prevalent in fields such as chemistry and electronics and in areas of classical mechanics such as astronomy and molecular dynamics.

We will also be interested in a more general class of problems for which y depends on additional independent variables $x_1, x_2, \ldots, x_k$ (k being the spatial dimension) and in which the function f depends on the partial derivatives of y with respect to these independent variables:

$$\frac{\partial y}{\partial t} = f(t, y, \frac{\partial y}{\partial x_1}, \frac{\partial y}{\partial x_2}, \ldots, \frac{\partial^2 y}{\partial x_1^2}, \frac{\partial^2 y}{\partial x_1 \partial x_2}, \ldots). \tag{1.3}$$

Such partial differential equations (PDEs) arise in continuum mechanics, such as fluid dynamics, structural analysis and heat transfer. In practice, a commonly used technique is to replace the spatial derivatives in (1.3) with discrete approximations via, for example, the finite volume method, leaving an ODE (1.2). This technique is called the *method of lines* or *semi-discretization* method. In this case the dimension d of (1.2) frequently scales with a small grid parameter h according to $d \sim h^{-k}$, often resulting in scalable problems of very large dimension. Some parts of this dissertation will deal with efficient methods for integrating the incompressible Navier-Stokes equations of fluid dynamics.

Finally, a somewhat more general class of methods than (1.2) is the class of differential algebraic equations (DAEs), of which the simplest example is a constrained ODE:

$$
\begin{aligned}
&y'(t) = f(t, y(t)), && (1.4)\\
&g(y) = 0, && (1.5)\\
&t \in \mathbb{R},\ y : \mathbb{R} \to \mathbb{R}^d, &&\\
&y(0) = y_0,\ g(y_0) = 0 &&
\end{aligned}
$$

In this case one wishes to find the solution of (1.4) on the constraint manifold defined by (1.5). The existence of an algebraic constraint can introduce additional complexity into the numerical method. In fluid flows, the condition of incompressibility can be seen as a constraint, and the pressure correction method, discussed later, can be seen as a projection onto the constraint manifold: the space of divergence-free vector fields.

1.1.1 Integration methods for differential equations

Many numerical integration methods approximate the solution of (1.2) over a step of size τ, by a mapping from $y_n \approx y(t_n)$ to $y_{n+1} \approx y(t_{n+1})$, where $t_{n+1} = t_n + \tau$. The characteristic of a numerical integrator having the greatest consequences for efficiency is whether the mapping is an *explicit* or *implicit* one. We shall define an explicit method as one for which the numerical solution over a time step can be obtained in exact arithmetic within a finite number of operations, *independent of the structure of the function f* (except for the assumption that f can be evaluated in finite operations, of course). The simplest explicit method is the explicit Euler method

$$y_{n+1} = y_n + \tau f(t_n, y_n).$$

Under the assumption that f can be evaluated in a finite number of operations, the additional cost of this method is that of a scalar multiplication and vector addition.

An implicit method generally requires an iteration to obtain the solution. The simplest example is the implicit Euler method:

$$y_{n+1} = y_n + \tau f(t_{n+1}, y_{n+1}). \tag{1.6}$$

Since y_{n+1} is yet to be computed, f cannot be simply evaluated. An important special case is when f is linear

$$f(t, y) \equiv Jy,$$

where J is a $d \times d$ matrix. In this case we can solve (1.6) from

$$(I - \tau J)y_{n+1} = y_n \tag{1.7}$$

by solving a single linear system of dimension d. For general nonlinear f, however, an iteration such as modified Newton's method must be used to solve the implicit relations:

$$(I - \tau J(y_n))(y_{n+1}^{(j+1)} - y_{n+1}^{(j)}) = y_n - y_{n+1}^{(j)} - \tau f(t_{n+1}, y_{n+1}^{(j)}), \quad j = 0, 1, \ldots$$

where $J(y_n) = \partial f(y)/\partial y$ is the Jacobian of f evaluated at y_n. Lacking a better initial approximate, $y_{n+1}^{(0)} = y_n$ may be used. This iteration is carried out for increasing j until satisfactory convergence is achieved. In each iteration a linear system with matrix $I - \tau J(y_n)$ must be solved.

1.1.2 Efficiency of numerical integration methods

By comparing the explicit- and implicit-Euler methods, one can see that, in general, the cost of computing one timestep with an implicit method is much more expensive than with an explicit method. But what is the difference in production?

Explicit methods are usually less robust in the presence of (always present) errors in the numerical solution. There is typically a problem dependent range of stepsizes $0 \leq \tau < \tau_0$ within which small errors are damped away and outside of which small errors are amplified. Clearly these *stability considerations* place a limit on the maximum allowable step which can be used by explicit methods.

On the other hand, there exist implicit methods, such as implicit-Euler which are *unconditionally stable*; that is, errors are damped out for any stepsize. For application to a linear problem (1.7), once the matrix $I - \tau J$ is factorized, the cost of solving the implicit relations may be sufficiently cheap to make it competitive with an explicit method with overly restrictive stepsize limitation. What's more, there is a large class of applications whose solutions are composed of modes with extremely different time scales, for which the stepsizes of explicit methods are so restricted as to render them useless. These problems are known as *stiff*, and practically demand treatment by an implicit method.

Improving the efficiency of an explicit method is not easy. One should look for problem-specific simplifications due to the coupling structure of the dependent variables or to known relations between problem variables. Implicit methods leave more room for improvement: one may attempt to accelerate the Newton process, by using iterative solvers, for example, or one may try to make the strategies for controlling the error and convergence rate more efficient. Additionally one may seek methods which reduce the degree of inherent implicitness. In the next section we consider a technique which may be used for certain problems to improve efficiency through combination of implicit and explicit methods.

1.1.3 Implicit-explicit splittings

In cases where stiffness plays a role, one can sometimes identify a splitting of the function f of the form $f(t,y) = f_E(t,y) + f_I(t,y)$ into terms f_I requiring implicit treatment due to stiffness, and terms $f_E(y)$ which may be treated explicitly. The diffusion term encountered in the incompressible Navier-Stokes equations is stiff though linear, for example, while the nonlinear terms are nonstiff. In this and similar examples from atmospheric chemistry, it may be advantageous to use a method which treats part of the derivative implicitly and part explicitly. Such methods are termed *implicit-explicit* (IMEX) methods, and the simplest example is the IMEX-Euler method:

$$y_{n+1} = y_n + \tau f_E(t_n, y_n) + \tau f_I(t_{n+1}, y_{n+1})$$

By introducing such a method, one hopes to find a more easily solvable implicit relation. Note that in the serendipitous case that the implicit part is linear, $f_I(t,y) \equiv Jy$, the result is a linearly implicit system requiring just one linear system solve. The Navier-Stokes equations are an example of a system in which the stiff part is linear.

The IMEX-Euler scheme inherits the stability region of the explicit-Euler method; that is to say, the time step must be small enough for explicit-part stability. More accurate IMEX methods suffer an additional restriction on the maximum stepsize, however. This additional restriction is carefully considered for the class of implicit-explicit linear multistep methods in Chapter 2.

1.2 Parallel computing for differential equations

Whether one measures efficiency in flops or seconds is of lesser importance when considering algorithms for sequential computing than when parallel algorithms are the subject. A parallel program is usually less efficient than an equivalent sequential program if cost is measured in flops. However one hopes to gain an advantage in execution time (wall-clock time, waiting time) worth more than the loss in flops.

Gear [4] points out two orthogonal approaches to parallelization of ODE methods: 1) parallelism across the problem space and 2) parallelism across the method space. Recently a third approach has been the subject of much research: 3) parallelism across time. The first two of these methods will be considered in the following sections. In the third approach one attempts to compute different time levels on different processors. This includes the class of waveform relaxation methods; it will not be considered in this dissertation, but the reader is directed to [1], [6] for further information.

By *orthogonal*, we mean that, ideally speaking, these three approaches to parallelization of ODE methods may be used independently of each other, and that when combined the the speedups and processor usage of both methods are realized. The usefulness of this orthogonality will be discussed in the last section of this chapter.

1.2.1 Parallelism across the problem space

In this approach, the method is divided across the problem dimension d. As as simple example, assume that p processors are available and that $m = d/p$ is an integer. Each processor is responsible for m components of the solution and coupling encountered in the evaluation of derivative function and the Jacobian and in the solution of linear systems requires communication. Clearly for this to be efficient, m (and thus d) should be sufficiently large. An effective distribution of data should minimize inter-processor coupling and maintain load balancing, making this approach difficult to apply for general differential equations.

There is, however, an important class of problems for which this kind of parallelism is *very* effective; namely, the class of semi-discretized PDEs are usually of high dimension with sparse, local coupling, providing an almost automatic load balancing by simply dividing up the domain of interest into p connected subdomains. The communication costs are proportional to the length of the shared interface between blocks. Since communication costs grow at a much slower rate than the grid size, it is intuitive that methods based on such a distribution improve in parallel efficiency as d is increased for a fixed number of processors. Because the communication costs are relatively cheap, and the problem size is in principle very large, it is feasible to solve these problems on distributed memory machines and even workstation clusters.

Parallel linear system solvers

The solution of linear systems is by far the most computation intensive task of time-dependent PDE simulations. Since the systems are large and sparse, iterative methods are recommended, if not required. Modern iterative methods are Krylov subspace methods and multigrid methods. The most important tasks for parallelization are matrix-vector multiplications, inner products and preconditioners. Matrix-vector products with local coupling are easily implemented with exchange of interface data, and this scales well, since only nearest neighbor communications are required. Inner products are also easily parallelized, but since the communication required is global, these do not scale well. To overcome this, methods have been suggested where inner products are grouped together to reduce the number of global communications. Other methods try to overlap such communications with useful computations to alleviate the cost somewhat. But these costs are rarely the bottleneck. The choice of a preconditioner for parallel applications is a trade-off between communication and computation costs and convergence rate. Domain decomposition methods, to be discussed in the next section, offer a spectrum of parallel preconditioners.

In time integration of fluid flow problems, high accuracy is rarely demanded, and we will consider only the implicit Euler method when dealing with the incompressible Navier-Stokes equations in this dissertation. The results obtained there may be immediately generalized to linear multistep methods, however.

The incompressible Navier-Stokes equations are special in that they include a constraint. For discussing the issues related to time integration, the following model of the Navier-Stokes equations is sufficient:

$$\begin{aligned} y' &= A(y) + Gp, \\ 0 &= G^T y. \end{aligned}$$

The function $A(y)$ represents the (discrete) momentum operator, p the pressure, and G the discrete gradient operator. To apply implicit Euler to this system we would like to solve

$$y_{n+1} = y_n + \tau(A(y_{n+1}) + Gp_{n+1})$$

subject to the constraint $G^T y_{n+1} = 0$, from which it follows that

$$G^T G p_{n+1} = -G^T A(y_{n+1}),$$

assuming the solution y_n from the previous time step is also divergence-free. No global loss of accuracy occurs if we compute the pressure field using the velocity field obtained by an unconstrained Euler step, and subtract the divergence from that velocity field, providing the following three-step algorithm:

$$\begin{aligned} y^*_{n+1} &= y_n + \tau A(y^*_{n+1}) && (1.8) \\ G^T G p_{n+1} &= -G^T A(y^*_{n+1}) && (1.9) \\ y_{n+1} &= y^*_{n+1} + \tau G^T p_{n+1}. \end{aligned}$$

The matrix $G^T G$ is similar to the familiar discrete Laplacian matrix. Due to the poor conditioning of this system, especially for large d, its solution may require 50%–80% of the computation time of incompressible flow simulations. Solution of the nonlinear relation (1.8) is typically easier: it is well-conditioned and often may be approximated by a linear system without loss of order—but there are cases where the nonlinearity becomes important, requiring Newton iterations[5].

For a complete overview of parallel methods for solving linear systems of equations, see [2].

Parallel preconditioners

Parallel preconditioners are also reviewed in [2]. These can be roughly divided into two classes: methods which start with a classical sequential preconditioner and attempt to distribute the data, and domain decomposition methods which start with distributed data and attempt to build a preconditioner. In this dissertation we will consider the second class of methods; the interested reader is referred to [2] and the references therein for the first class (as well as for a more complete review of the second class).

An introduction to domain decomposition methods is discussed in Appendix A. In Chapter 3 we consider a nonoverlapping additive Schwarz preconditioner for the Poisson equation, in which the blocks are approximated by an inner iteration or simply by an incomplete factorization preconditioner. Since it requires no communication between blocks, this method is perfectly parallel. In Chapter 4 the same method is applied to the incompressible Navier-Stokes equations.

To make domain decomposition very effective as a parallel preconditioner, one should provide overlap of subdomains and should perform a coarse grid correction for global communication of the error. Particularly the implementation of overlap may require major code revisions if it is desired to introduce domain decomposition into a mature code with inflexible data structures.

1.2.2 Parallelism across the method space

Although second order accuracy in time is often sufficient for PDEs, higher order methods are needed for precise computations in fields such as electronics, chemistry and astronomy. These applications also frequently involve very complex nonlinear relations. The parallelism inherent in higher order methods is normally small scale, improving as the order is increased, and the tight coupling of the relations is best suited for implementation on shared memory computers.

Many of these methods fit into a class of general linear s-stage methods:

$$Y_{n+1} - \tau(A \otimes I)F(Y_{n+1}) = (G \otimes I)V_n, \; y_{n+1} = (b^T \otimes I)Y_{n+1}, \tag{1.10}$$

where

$$Y_{n+1} = \begin{pmatrix} y_{n+1}^1 \\ \vdots \\ y_{n+1}^s \end{pmatrix}, \; V_n = \begin{pmatrix} y_{n-k+1} \\ \vdots \\ y_n \end{pmatrix},$$

$y_{n+1}^i \in \mathbb{R}$, $A \in \mathbb{R}^{s\times s}$, $G \in \mathbb{R}^{s\times k}$, $b \in \mathbb{R}^s$ and $F(Y_{n+1}) = (f(y_{n+1}^1)^T, \ldots, f(y_{n+1}^s)^T)^T$. The implicit relations (1.10) may be solved with a Newton iteration:

$$(I - \tau A \otimes J^*)(Y_{n+1}^{(j)} - Y_{n+1}^{(j-1)}) = -R_n(Y_{n+1}^{(j-1)}), \tag{1.11}$$

where J^* is an appropriate approximation to the Jacobian and the residue $R_n(Y)$ is defined by

$$R_n(Y) = Y - \tau(A \otimes I)F(Y) - (G \otimes I)V_n.$$

The matrix A above is normally full, but if it is nondefective, there is a matrix Q such that $Q^{-1}AQ = D$, where D is a diagonal matrix. In this case it is possible to perform an equivalent, transformed Newton iteration

$$\begin{aligned} (I - \tau D \otimes J^*)(X_{n+1}^{(j)} - X_{n+1}^{(j-1)}) &= -(Q^{-1} \otimes I)R((Q \otimes I)X_{n+1}^{(j-1)}), \\ Y_{n+1} &= (Q \otimes I)X_{n+1}, \end{aligned} \tag{1.12}$$

comprised of s decoupled linear systems with matrices $(I - \tau d_i J^*)$, $i = 1, \ldots, s$, where the d_i are the diagonal entries of D.

Note that there is a lot of parallelism available in (1.12). At the coarsest level, the s blocks of $(I - \tau D \otimes J^*)$ can be factored and solved solved independently of each other for the stage updates. In a similar manner the stages of F can be computed in parallel. If, in addition, all stages are available to all processors, the residue function and the transformation can be computed concurrently. Fine-grain parallelism is also plentiful: all vector updates can be parallelized component-wise, and the individual components of the derivative function f and the Jacobian J can be computed simultaneously, though these operations require more careful load-balancing and more efficient memory management than is commonly available on current architectures.

If A is not diagonalizable, or if its eigenvalues are complex, then an alternative must be sought. A class of methods based on this idea is developed and analyzed in Chapters 5 and 6.

It is clear that parallelism across the method space in (1.12) is restricted to a number of processors equal to the number of stages s. However, a large number of stages is only needed if high order accuracy is required. It follows that parallelism across the method space is most beneficial when high accuracy in time is desired.

1.2.3 Parallel (||) "orthogonality(⊥)"?

From the preceding discussions we can identify the classes of problems for which the two parallelization approaches discussed at the beginning of Section 1.2 are best suited:

- *Parallelism across the method parameters*—The amount of parallelism is proportional to the number of stages of the method. Since more stages are normally associated with greater accuracy, we are looking for applications requiring high order. Furthermore, data transfer on the order of the problem dimension is expected in each timestep, so that shared memory is advantageous.
- *Parallelism across the problem space*—A good load balancing requires either knowledge of the connectivity graph or approximately equal connectivity between components. To reduce communications, the connectivity should be sparse. In this case distributed memory computers may be used.

While it is possible for tightly coupled dynamics applications requiring high accuracy (and thus allowing effective use of parallelization across the method parameters) to also take advantage of parallelism across the problem space by dividing up the vector sums, function evaluations and Jacobian evaluations into blocks, this will only be efficient on todays computers for fairly large problems.

Techniques exploiting parallelism across the problem space are ideally suited for application to time dependent PDEs on massively parallel distributed memory computers, but in general, high order is not needed in PDE applications, where the accuracy of the spatial discretization is normally the limiting factor. The orthogonality of the parallelisms across space and across the method is in this sense overrated.

References

[1] K. Burrage. *Parallel and Sequential Methods for Ordinary Differential Equations.* Oxford University Press, 1995.

[2] J.J. Dongarra, I.S. Duff, D.C.Sorensen, and H.A. van der Vorst. *Numerical Linear Algebra for High-Performance Computers.* SIAM, Philadelphia, 1998.

[3] Ed. F.C. Mish. *Merriam Webster's Collegiate Dictionary, Tenth Edition.* Merriam-Webster, Incorporated, Springfield, Massachusetts, 1998.

[4] C.W. Gear. Parallel methods for ordinary differenital equations. *Calcolo*, 25:1–20, 1988.

[5] D. R. van der Heul, C. Vuik, and P. Wesseling. Stability analysis of segregated solution methods for compressible fluid flows. Preprint, submitted for publication, 1999.

[6] S. Vandewalle. *Parallel multigrid waveform relaxation for parabolic problems.* B. G. Teubner Verlag, Stuttgart, 1993.

Chapter 2

On the Stability of Implicit-Explicit Linear Multistep Methods

Abstract. In many applications, such as atmospheric chemistry, large systems of ordinary differential equations (ODEs) with both stiff and nonstiff parts have to be solved numerically. A popular approach in such cases is to integrate the stiff parts implicitly and the nonstiff parts explicitly. In this paper we study a class of implicit-explicit (IMEX) linear multistep methods intended for such applications. The paper focuses on the linear stability of popular second order methods like extrapolated BDF, Crank-Nicolson Leap-Frog and a particular class of Adams methods. We present results for problems with decoupled eigenvalues and comment on some specific CFL restrictions associated with advection terms.

2.1 Implicit-explicit linear multistep methods

When adopting the method of lines approach, space discretization of multi-space dimensional, time dependent PDE problems results in large systems of ODEs which are to be integrated in time by an appropriate time stepping scheme. Frequently in such applications one is confronted with problems having both stiff and nonstiff parts. Here the term nonstiff is used in a loose way to indicate terms that may be solved efficiently in an explicit way. For example, in atmospheric chemistry one may have a nonstiff horizontal advection term and a stiff term containing chemical reactions and vertical diffusion, see for instance Verwer et al. [10], Zlatev [12]. In such cases it is desirable to treat the stiff part with an implicit scheme while applying an explicit scheme to the nonstiff part.

In this paper we look at the general ODE problem

$$w'(t) = F(t, w(t)) + G(t, w(t)), \qquad t \geq 0, \tag{2.1}$$

where F represents the nonstiff part and G represents the stiff part of the system. For the numerical solution of (2.1) we consider *implicit-explicit* (IMEX) linear multistep methods

$$\sum_{j=0}^{k} a_j w_{n+1-j} = \tau \sum_{j=1}^{k} b_j F(t_{n+1-j}, w_{n+1-j}) + \tau \sum_{j=0}^{k} c_j G(t_{n+1-j}, w_{n+1-j}). \tag{2.2}$$

Here $\tau > 0$ denotes the time step and the vectors w_n approximate the exact solution at $t_n = n\tau$. Schemes of this type were introduced by Crouzeix [3] and Varah [9].

A natural way to derive such a method is to start with an implicit method that is known to possess favorable stability properties, and then replace the term $F(t_{n+1}, w_{n+1})$ by a linear combination of explicit terms using extrapolation. If the implicit method has order p and the extrapolation is of order q, the resulting scheme will be of order $\min\{p, q\}$, see [6]. On the other hand, it is not hard to see from the proof of [6] that any consistent IMEX linear multistep method can be decomposed into an implicit scheme and an extrapolation procedure. Direct derivations of the order conditions for IMEX linear multistep methods are given in Ascher et al. [2].

In this paper we will discuss the stability properties of the schemes for the scalar, complex test equation

$$w'(t) = \lambda w(t) + \mu w(t). \tag{2.3}$$

In applications for PDEs, these λ and μ represent the eigenvalues of the nonstiff and stiff part, respectively, found by a Fourier analysis. We will not assume that λ and μ are coupled, so that F and G may contain discretized spatial derivatives in different directions. To simplify the notation, we will make in the following the substitutions $\lambda \to \tau\lambda$ and $\mu \to \tau\mu$. Application of the IMEX scheme then gives

$$\sum_{j=0}^{k} a_j w_{n+1-j} = \lambda \sum_{j=1}^{k} b_j w_{n+1-j} + \mu \sum_{j=0}^{k} c_j w_{n+1-j}. \tag{2.4}$$

As a simple example consider the first order IMEX Euler method

$$w_{n+1} - w_n = \tau F(t_n, w_n) + \tau G(t_{n+1}, w_{n+1}). \tag{2.5}$$

For the linear test equation this gives

$$w_{n+1} = (1-\mu)^{-1}(1+\lambda)w_n,$$

and it easily follows that the method is stable whenever λ lies in the stability region of the explicit Euler method, $|1+\lambda| \leq 1$, and μ is in the stability region of the implicit Euler method, $|1-\mu| \geq 1$. As we shall see, this is an exceptional situation. Usually, stability of the individual explicit and implicit methods does not guarantee stability of the combined IMEX method.

In this paper we consider several second order methods, where the implicit method is A-stable. We shall address two questions:

- Suppose that λ lies in the stability region $\mathcal{S}$ of the explicit method. What restrictions are to be placed on the location of μ to have stability?
- What additional restrictions, if any, are to be imposed on the location of λ to ensure that the method is stable for all μ in the left half-plane?

Some examples of IMEX methods that seem interesting for practical applications are given in Section 2. In Section 3 we discuss the restrictions on λ for having stability for arbitrary μ in the left half-plane. In Section 4 we discuss the question of stability of the IMEX methods under the assumption that λ lies in the stability region of the explicit method. Some consequences for CFL restrictions are considered in Section 5, where λ will be an eigenvalue for advection discretizations.

Related stability results for IMEX multistep methods have been derived by Varah [9] and Ascher et al. [2] for the one-dimensional convection-diffusion problem, with central spatial discretizations, where the convection is treated explicitly. For such problems there will be a coupling between the eigenvalues λ and μ. The results presented in this paper are applicable to more general problems and more general spatial discretizations, since λ and μ are considered to be independent of each other and the specific form of the eigenvalues is not prescribed a priori.

Clearly there is a big gap between the test equation (2.3) and the general problem (2.1). Results for (2.3) can be easily extended to linear systems with normal, commuting matrices. Note that if $F = L \otimes I$ and $G = I \otimes M$, with Kronecker product $\otimes$, then F and G will commute. Matrices of this type arise from linear PDE problems with constant coefficients if F and G contain discretized spatial derivatives in different directions. Stability and convergence results for the noncommuting case, but where G is assumed to be negative definite, can be found in Crouzeix [3]. Generalizations for G linear, negative definite and F nonlinear are given in Akrivis et al. [1]. Here we shall restrict ourselves to the scalar, linear case but there is no a priori restriction on the location of the eigenvalues λ and μ other than that they should lie in the stability region of the explicit or implicit multistep method, respectively.

2.2 Preliminaries

Stability of (2.4) is determined by the location of the roots of the characteristic equation

$$\sum_{i=0}^{k} a_i \zeta^{k-i} - \lambda \sum_{i=1}^{k} b_i \zeta^{k-i} - \mu \sum_{i=0}^{k} c_i \zeta^{k-i} = 0. \tag{2.6}$$

For a root ζ, stability requires that $|\zeta| \leq 1$, with strict inequality for multiple roots, see for instance [4, 5, 7]. If this last condition is omitted, a weak, polynomial instability may occur. The requirement that $|\zeta| \leq 1$ is more important, since its violation will lead to an exponential blow-up.

Dividing the equation by ζ^k and making the substitution $z = 1/\zeta$, the characteristic equation reads

$$A(z) - \lambda B(z) - \mu C(z) = 0,$$

where A, B and C are the polynomials

$$A(z) = \sum_{i=0}^{k} a_i z^i, \quad B(z) = \sum_{i=1}^{k} b_i z^i, \quad C(z) = \sum_{i=0}^{k} c_i z^i. \tag{2.7}$$

So, for stability we require that all roots satisfy $|z| \geq 1$, again with strict inequality if z is a multiple root. A necessary condition for this is

$$A(z) - \lambda B(z) - \mu C(z) \neq 0 \quad \text{for all } |z| < 1. \tag{2.8}$$

Apart from the possibility of multiple roots with modulus 1 this is also a sufficient condition. We shall use (2.8) as a criterion for determining stability. On the boundaries of the stability domains it can then be verified separately whether multiple roots with modulus 1 occur.

In the following we denote by $\mathcal{S}$ the stability region of the explicit method. Its interior $\mathrm{int}(\mathcal{S})$, where all characteristic roots have modulus less than 1, is given by the *complement* of the set $\{A(z)/B(z) : |z| \leq 1\}$, as can be seen from the above by setting $\mu = 0$. The boundary of the stability region is contained in the root locus curve

$$\{A(e^{i\theta})/B(e^{i\theta}) : \theta \in [-\pi, \pi]\}. \tag{2.9}$$

Below we give some examples of IMEX multistep methods with the stability regions of the explicit methods. The attention will be restricted to second order methods for which the implicit method is A-stable. We shall denote $F_n = F(t_n, w_n)$ and $G_n = G(t_n, w_n)$.

EXAMPLE 2.1 (Crank-Nicolson Leap-Frog). Using the explicit midpoint method (Leap-Frog) for the explicit part with trapezoidal rule (Crank-Nicolson) for the implicit part provides the popular scheme

$$\frac{1}{2}(w_{n+1} - w_{n-1}) = \tau F_n + \frac{1}{2}\tau(G_{n+1} + G_{n-1}). \tag{2.10}$$

The polynomials (2.7) for this method are

$$A(z) = \frac{1}{2}(1 - z^2), \quad B(z) = z, \quad C(z) = \frac{1}{2}(1 + z^2),$$

and the root locus curve for the stability region of the explicit method is

$$\lambda(\theta) = \frac{1 - e^{i2\theta}}{2e^{i\theta}} = -i \sin\theta, \qquad \theta \in [-\pi, \pi].$$

That is, the explicit eigenvalues λ must be restricted to the imaginary axis between $-i$ and i. For the extremal values $\lambda = \pm i$ the roots of the characteristic equation coincide, so we then have a linear instability.

EXAMPLE 2.2 (Extrapolated BDF). A second order IMEX method can be derived from the two-step backward differentiation formula, with extrapolation $F_{n+1} \approx 2F_n - F_{n-1}$ for the explicit part, thus giving

$$\frac{3}{2}w_{n+1} - 2w_n + \frac{1}{2}w_{n-1} = \tau(2F_n - F_{n-1}) + \tau G_{n+1}. \tag{2.11}$$

In Verwer et al. [10] this method was applied successfully to a large system (2.1) arising from spatial discretization of an atmospheric transport-chemistry model. The implementation there was slightly different, with $F(2w_n - w_{n-1})$ instead of $2F_n - F_{n-1}$, but for linear stability this is irrelevant.

The polynomials (2.7) are given by

$$A(z) = \frac{1}{2}(3-z)(1-z), \quad B(z) = z(2-z), \quad C(z) = 1,$$

and the boundary of the explicit stability region $\mathcal{S}$ is parameterized by

$$\lambda(\theta) = \frac{(3-e^{i\theta})(1-e^{i\theta})}{2e^{i\theta}(2-e^{i\theta})}, \qquad \theta \in [-\pi, \pi].$$

EXAMPLE 2.3 (Adams methods). We consider the class of second order Adams type methods, with parameter $c \geq 0$,

$$w_{n+1} - w_n = \frac{3}{2}\tau F_n - \frac{1}{2}\tau F_{n-1} + \frac{1}{2}(1+c)\tau G_{n+1} + \frac{1}{2}(1-2c)\tau G_n + \frac{1}{2}c\tau G_{n-1}. \tag{2.12}$$

Again these methods can be obtained from the implicit formula by extrapolation. The implicit methods are A-stable for any $c \geq 0$. For $c = 0$ the implicit method is simply the trapezoidal rule (Crank-Nicolson). The choice $c = 1/8$ was considered by Ascher et al. [2]; within this class $c = 1/8$ yields maximal damping at $\mu = \infty$. The implicit method with $c = 1/2$ was advocated by Nevanlinna and Liniger [8], with regard to maximum norm contractivity.

The polynomials (2.7) are given by

$$A(z) = 1 - z, \quad B(z) = \frac{1}{2}z(3-z), \quad C(z) = \frac{1}{2}(1+c) + \frac{1}{2}(1-2c)z + \frac{1}{2}cz^2.$$

The boundary of the stability region of the explicit method, the two-step Adams-Bashforth method, is given by

$$\lambda(\theta) = \frac{2(1-e^{i\theta})}{e^{i\theta}(3-e^{i\theta})}, \qquad \theta \in [-\pi, \pi].$$

2.3 Restrictions on explicit eigenvalues for implicit A-stability

Defining $\varphi_\lambda(z) = (A(z) - \lambda B(z))/C(z)$, criterion (2.8) reads

$$\mu \neq \varphi_\lambda(z) \quad \text{for any } |z| < 1. \tag{2.13}$$

We shall apply this criterion to determine under what conditions we have A-stability with respect to the implicit eigenvalue, that is stability for arbitrary $\mu \in \mathcal{C}^-$, the left half-plane. In the first section it was noted already that the IMEX Euler method remains A-stable with respect to the implicit eigenvalues so long as all of the explicit eigenvalues are in the stability region of the explicit method. We can show a similar result for the Crank-Nicolson Leap-Frog scheme (2.10).

Example 3.1. For scheme (2.10), with $\lambda = -i \sin\theta$ in the explicit stability region, we have

$$\varphi_\lambda(z) = \frac{1 - z^2 + 2zi\sin\theta}{1 + z^2} = \frac{1 - (z^2 - \bar{z}^2) - |z|^4 + 2iz\sin\theta(1 + \bar{z}^2)}{1 + (z^2 + \bar{z}^2) + |z|^4}.$$

The denominator of this last expression is obviously positive for $|z| < 1$. The real part of the numerator is

$$1 - |z|^4 - 2\sin\theta \operatorname{Im}\left[z(1 + \bar{z}^2)\right] = (1 - |z|^2)\left[1 + x^2 + y^2 - 2y\sin\theta\right] > 0$$

for any $|z| < 1$, $z = x + iy$. So A-stability for the implicit eigenvalues μ is preserved as long as the explicit eigenvalues λ are in $\mathcal{S}$.

As we shall see, for the other methods of Examples 2.2 and 2.3 A-stability for the implicit eigenvalues is not preserved for arbitrary $\lambda \in \mathcal{S}$. We define

$$\mathcal{D} = \{\lambda \in \mathbb{C} : \text{ (2.6) is stable for any } \mu \in \mathbb{C}^-\}. \tag{2.14}$$

Obviously, $\mathcal{D}$ will be a subset of the closure of the explicit stability domain $\mathcal{S}$. The following lemma gives a characterization for the boundary in terms of the functions $M(\theta) = A(e^{i\theta})/C(e^{i\theta})$ and $N(\theta) = B(e^{i\theta})/C(e^{i\theta})$.

Lemma 3.2. Suppose $\operatorname{Re}N(\theta) \not\equiv 0$ and $M(\theta), N(\theta)$ are bounded for all $\theta \in [-\pi, \pi]$. Then $\partial\mathcal{D} \subset \{\lambda(\theta) : \theta \in [-\pi, \pi]\}$ with

$$\lambda(\theta) = \frac{d}{d\theta}\left(\frac{M(\theta) + M(-\theta)}{N(-\theta)}\right)\left[\frac{d}{d\theta}\left(\frac{N(\theta)}{N(-\theta)}\right)\right]^{-1}. \tag{2.15}$$

Proof. If $\lambda \in \mathcal{D}$ then, according to (2.13), φ_λ maps the interior of the unit disc into the right half-plane. By assumption the image of the unit disc under φ_λ is bounded. For a point on the boundary of $\mathcal{D}$ we thus have

$$\operatorname{Re}\varphi_\lambda(e^{i\theta}) = 0 \tag{2.16}$$

for some point $e^{i\theta}$ on the unit circle. Moreover,

$$\frac{d}{d\theta}\mathrm{Re}\,\varphi_\lambda(e^{i\theta}) = 0, \tag{2.17}$$

which is necessary so that $\mathrm{Re}\,\varphi_\lambda(e^{i\theta})$ does not become negative for points near $z = e^{i\theta}$ on the unit disk. We can simplify these conditions somewhat, obtaining a single parameterization in terms of the functions M and N, evaluated in the point θ, as follows:

$$\begin{cases} 0 &= (\bar{M} - \bar{N}\bar{\lambda}) + (M - N\lambda), \\ 0 &= (\bar{M}' - \bar{N}'\bar{\lambda}) + (M' - N'\lambda). \end{cases}$$

Solving this system for $\lambda = \lambda(\theta)$, we obtain

$$\lambda(\theta) = \frac{-\bar{N}\bar{M}' + \bar{N}'\bar{M} - \bar{N}M' + \bar{N}'M}{\bar{N}'N - \bar{N}N'} = \frac{\bar{N}(\bar{M}' + M') - \bar{N}'(\bar{M} + M)}{\bar{N}N' - \bar{N}'N},$$

which is equivalent to (2.16). This expression is well defined iff $N(\theta)$ is not identically equal to $\overline{N(\theta)}$. □

For specific methods the boundary of the set $\mathcal{D}$ can be parameterized by evaluating (2.15) for $\theta \in [-\pi, \pi]$. For the IMEX-BDF method (2.11) this leads to a region $\mathcal{D}$ with boundary

$$\lambda(\theta) = -\frac{1}{6}(1 - e^{i\theta})(3 - e^{i\theta}), \tag{2.18}$$

see Figure 2.1. This region seems only marginally smaller than the explicit stability region $\mathcal{S}$. Note however that near the origin $\mathcal{S}$ stays closer to the imaginary axis than $\mathcal{D}$.

For the IMEX-Adams schemes (2.7) we get the more complicated formula, found by Maple,

$$\lambda(\theta) = P(e^{i\theta})/Q(e^{i\theta}) \tag{2.19}$$

with

$$\begin{aligned} P(z) = {}& c\,(z-1)\Big(-6cz^3 + 2(5c-6)z^2 - 2(c+12)z^2 - 2(c-2)\Big), \\ Q(z) = {}& 3(c+c^2)z^4 + 2(3-c-4c^2)z^3 + 2(6+11c+5c^2)z^2 + \\ & +2(3-c-4c^2)z + 3(c+c^2). \end{aligned} \tag{2.20}$$

The $\mathcal{D}$ regions for $c = 1/8$ and $c = 1/2$ are given in Figure 2.2. For $c = 1/2$ it is close to $\mathcal{S}$, whereas for $c = 1/8$ we lose a considerable part of the explicit stability region. For $c = 0$ the lemma does not apply since M and N are not bounded near $\theta = \pi$, and so we consider this method separately.

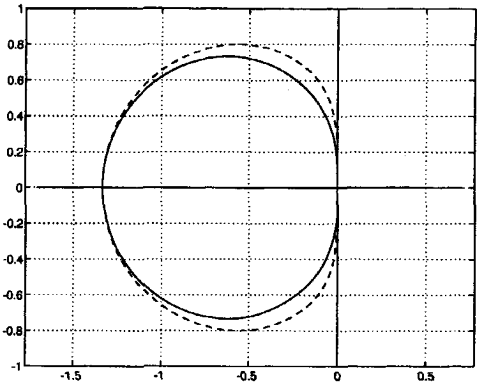

FIGURE 2.1: The explicit stability region $\mathcal{S}$ (dashed) and the region $\mathcal{D}$ for the IMEX-BDF2 method.

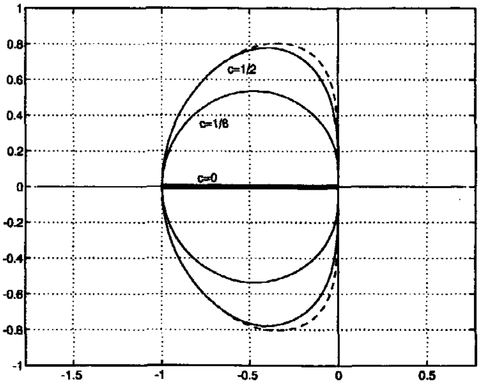

FIGURE 2.2: The explicit stability region $\mathcal{S}$ (dashed) and the regions $\mathcal{D}$ for the IMEX-Adams2 methods with $c = 1/2$, $1/8$ and 0.

EXAMPLE 3.3. For (2.12) with $c = 0$, the Adams-Bashforth Crank-Nicolson method, we have

$$\varphi_\lambda(z) = \frac{2(1-z) - \lambda z(3-z)}{1+z}.$$

This can be written as

$$\varphi_\lambda(z) = 2 - z + (1+\lambda)\chi(z) \qquad \text{with} \qquad \chi(z) = -\frac{z(3-z)}{1+z}. \tag{2.21}$$

By some calculations it follows that $\operatorname{Re}\chi(e^{i\theta}) = -2 + \cos\theta$. Note that for $\theta \to \pi$ the real part of $\chi(e^{i\theta})$ tends to -3 and its modulus to ∞. Hence χ maps the unit disk into the half plane $\{\zeta \in \mathbb{C} : \operatorname{Re}\zeta \geq -3\}$ and the imaginary axis lies totally in this image. It follows that the image of the unit disk under φ_λ will have a nonempty intersection with the left half plane if $1 + \lambda$ has a nonzero imaginary part. Therefore λ has to be real to be in $\mathcal{D}$.

Since $\mathcal{D}$ is a subset of $\mathcal{S}$, the only possible values are in the interval $[-1, 0]$. Indeed any λ on this piece of the real negative axis is in $\mathcal{D}$. This can be seen as follows: we have for real λ

$$\operatorname{Re}\varphi_\lambda(e^{i\theta}) = -\lambda(2 - \cos\theta) \geq 0 \qquad \text{if } \lambda \leq 0,$$

and from (2.21) it now follows that the unit disk is mapped into the right half-plane if $\lambda \leq 0$ and $1 + \lambda \geq 0$.

2.4 Restrictions on implicit eigenvalues for full explicit stability

Although A-stability is a valuable property, in most practical situations one can settle for less demanding properties, such as $A(\alpha)$-stability. In this section we consider what requirements on μ are needed to ensure stability of the IMEX methods for arbitrary $\lambda \in \mathcal{S}$. The implicit eigenvalues μ are supposed to be in the wedge

$$\mathcal{W}_\alpha = \{\zeta \in \mathbb{C} : \ |\arg(-\zeta)| < \alpha\}$$

with angle $\alpha \in (0, \frac{1}{2}\pi)$.

LEMMA 4.1. Suppose that $|\arg(A(z) - \lambda B(z))| \leq \frac{1}{2}\pi + \beta$, $|\arg(C(z))| \leq \gamma$ for all $|z| = 1$ and $\lambda \in \partial\mathcal{S}$, with $\beta + \gamma < \frac{1}{2}\pi$. Then the IMEX scheme will be stable for any $\lambda \in \mathcal{S}$ and $\mu \in \mathcal{W}_\alpha$, with $\alpha = \frac{1}{2}\pi - \beta - \gamma$.

PROOF. We have

$$|\arg(\varphi_\lambda(z))| \leq |\arg(A(z) - \lambda B(z))| + |\arg(C(z))|.$$

From the assumptions it follows that $|\arg(\varphi_\lambda(z))| \le \frac{1}{2}\pi + \beta + \gamma$ for all $|z| \le 1$ and $\lambda \in \mathcal{S}$. Using criterion (2.13), the result follows. □

To determine the angle β in the above lemma for the 2-step methods, note that we can write

$$A(z) - \lambda B(z) = A(0)(1 - \rho_1 z)(1 - \rho_2 z),$$

where ρ_1 and ρ_2 are the characteristic roots of the explicit method. For $\lambda \in \partial\mathcal{S}$ we get $|\rho_1| = 1$ and $|\rho_2| \le r$ with some constant $r \le 1$ determined by the explicit method. It follows by geometrical considerations that we can take $\beta = \arcsin r$.

Example 4.2. Consider the IMEX-BDF2 scheme(2.11). The characteristic equation of the explicit method reads

$$\frac{3}{2}\zeta^2 - 2(1+\lambda)\zeta + \frac{1}{2}(1+2\lambda) = 0,$$

see (2.6) with $\mu = 0$. If $\lambda \in \partial\mathcal{S}$ we can set $\rho_1 = e^{i\theta}$ and by some calculations it is seen that

$$\rho_2 = \frac{1}{3}\frac{3e^{i\theta} - 2}{2e^{i\theta} - 1}, \qquad |\rho_2| \le \frac{5}{9}.$$

Further we have $\arg(C(z)) = 0$. Therefore Lemma 4.1 gives stability with angle

$$\alpha = \frac{\pi}{2} - \arcsin(\frac{5}{9}) \approx 0.31\pi. \tag{2.22}$$

The region of those μ for which we have stability with arbitrary $\lambda \in \mathcal{S}$ is given by the *complement* of the set $\{\varphi_\lambda(z) : \lambda \in \mathcal{S}, |z| < 1\}$. Although we do not have a parameterization of the boundary of this set, we can make a (crude) picture of it by plotting the value of $\varphi_\lambda(z)$ for sufficiently many $\lambda \in \mathcal{S}$ and $|z| < 1$. In Figure 2.3 this region is shown for the IMEX-BDF2 scheme. By zooming in on the origin one can establish an *experimental bound* of the angle α, and for this method it was found that $\alpha \approx 0.32\pi$, which is close to the lower bound (2.22).

Example 4.3. Consider the IMEX-Adams scheme (2.12). For the 2-step Adams-Bashforth method, with $\lambda \in \partial\mathcal{S}$, we get, similar to the previous example, $\rho_1 = e^{i\theta}$ and

$$\rho_2 = \frac{e^{i\theta} - 1}{3e^{i\theta} - 1}, \qquad |\rho_2| \le \frac{1}{2},$$

giving $\beta = \arcsin(1/2)$.

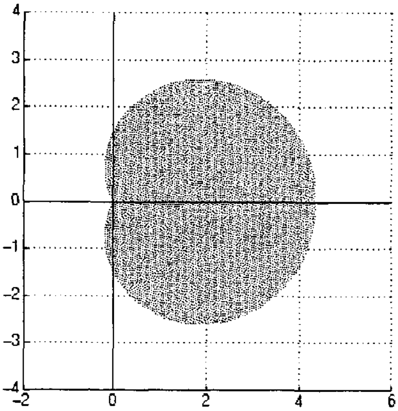

FIGURE 2.3: Exterior of shaded region: stability for μ with arbitrary $\lambda \in \mathcal{S}$ for the IMEX-BDF2 method.

If $c = 1/2$ then $C(z) = \frac{3}{4}(1 + \frac{1}{3}z^2)$, and thus we can take $\gamma = \arcsin(1/3)$. This gives stability of scheme (2.12) for $\lambda \in \partial\mathcal{S}$ and $\mu \in \mathcal{W}_\alpha$ with

$$\alpha = \frac{\pi}{2} - \arcsin(\frac{1}{2}) - \arcsin(\frac{1}{3}) \approx 0.23\pi. \tag{2.23}$$

As in the previous example we determined from Figure 2.4 an experimental bound for α and this was found to be $\approx 0.30\pi$, so here the lower bound (2.23) seems not very close.

If $c = 1/8$ then $C(z) = \frac{3}{16}(1 + \frac{1}{3}z)^2$, leading to $\gamma = 2\arcsin(1/3)$. This gives an angle

$$\alpha = \frac{\pi}{2} - \arcsin(\frac{1}{2}) - 2\arcsin(\frac{1}{3}) \approx 0.12\pi. \tag{2.24}$$

The experimental bound for this method, see Figure 2.4, was found to be $\approx 0.14\pi$.

EXAMPLE 4.4. For the IMEX-Adams scheme (2.12) with $c = 0$ we have $C(z) = \frac{1}{2}(1 + z)$, leading to $\gamma = \frac{1}{2}\pi$. Hence for his case Lemma 4.1 does not provide a positive angle α. Note that Lemma 4.1 only gives a sufficient condition. We show that indeed there is no positive α such that the scheme is stable for all $\lambda \in \mathcal{S}$ and $\mu \in \mathcal{W}_\alpha$.

We have

$$\varphi_\lambda(z) = \frac{2(1-z) - \lambda z(3-z)}{1+z}.$$

Now take $z = -1 + i\varepsilon + \mathcal{O}(\varepsilon^2)$ on the unit circle and $\lambda = -1 - i\varepsilon + \mathcal{O}(\varepsilon^2)$ on the boundary of $\mathcal{S}$, see Figure 2.2. Then $\varphi_\lambda(z) = -1 + \mathcal{O}(\varepsilon)$, showing that we can have values for $\varphi_\lambda(z)$ arbitrarily close to the negative real axis.

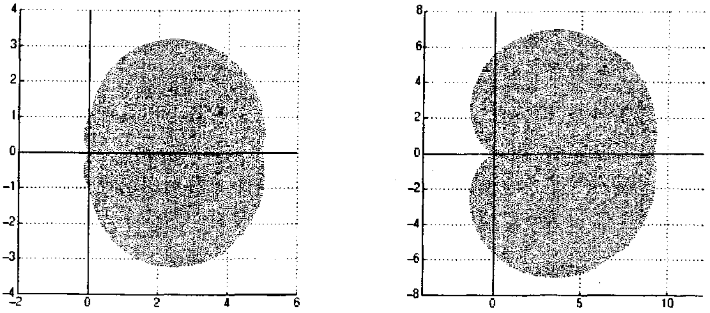

FIGURE 2.4: Exterior of shaded region: stability for μ with arbitrary $\lambda \in \mathcal{S}$ for the IMEX-Adams method with $c = 1/2$ (left) and $c = 1/8$ (right).

2.5 CFL restrictions for advection terms

So far we have followed an ODE stability analysis in the sense that the eigenvalues λ and μ were allowed to take on arbitrary complex values in certain bounded or unbounded regions in the complex plane. Of course, in actual applications they are determined by specific spatial operators and selected spatial discretization techniques. Often, the nonstiff part F in (1.1) emanates from advection and the stiff part G from reaction-diffusion terms. For example, in the study of atmospheric transport-chemistry models, a useful test model is the system $c_t + uc_x + vc_y = \epsilon c_{zz} + g(t, c)$, where c is a vector of concentrations, $uc_x + vc_y$ models advection in a horizontal wind field, ϵc_{zz} a vertical turbulent/diffusion process, and $g(t, c)$ stiff chemical reactions, see [10] for instance.

In this section we consider the specific case that λ is associated to the advection term uc_x while μ may still take on arbitrary values. We consider the first and third order upwind biased schemes for discretization on a uniform grid with grid size Δx. Let ν denote the Courant number $|u|\tau/\Delta x$. Then for the first order method we have explicit eigenvalues

$$\lambda = -\nu\Big(1 - \cos\theta + i\,\sin\theta\Big), \quad -\pi \leq \theta \leq \pi, \tag{2.25}$$

whereas in the third order case

$$\lambda = -\frac{1}{3}\nu\Big((\cos\theta - 1)^2 + i\,\sin\theta\,(4 - \cos\theta)\Big), \quad -\pi \leq \theta \leq \pi, \tag{2.26}$$

see for instance [10, 11]. Central advection discretization of even order leads to purely imaginary eigenvalues, and among the explicit methods considered here only the Leap-Frog method (2.10) will be stable.

We consider the restrictions on the Courant numbers ν for all explicit eigenvalues to be in the regions $\mathcal{S}$ or $\mathcal{D}$, introduced in Section 3. The bounds, given in Table 5.1, have been established experimentally.

	$\mathcal{S}$	$\mathcal{D}$	$\mathcal{S}$	$\mathcal{D}$	$\mathcal{S}$	$\mathcal{D}$
(2.25)	0.66	0.66	0.50	0.50	0.50	0.50
(2.26)	0.46	0.23	0.58	0.16	0.58	0.43
	IMEX-BDF2		Adams, $c = 1/8$		Adams, $c = 1/2$	

TABLE 5.1. CFL restrictions for the IMEX methods (2.11) and (2.12).

For applications, the results for the third order upwind discretizations seem more important than for the first order discretization. It is interesting to note the effect of the apparently moderate restriction for implicit A-stability of the IMEX-BDF2 method on the Courant number. If we demand A-stability for the stiff eigenvalues, the slightly smaller region $\mathcal{D}$ in Figure 2.1 results in a reduction of the maximal Courant number by approximately half. The reason for this is that eigenvalues of the third order upwind scheme are very close to the imaginary axis near zero. In this respect, among the IMEX schemes considered here, the Adams scheme (2.12) with $c = 1/2$ gives the best results. However, for practical purposes the results of Section 4 seem more important, and there the largest angle α was obtained for the BDF scheme (2.11).

In conclusion, both the IMEX-BDF method (2.11) and the IMEX-Adams method (2.12) with $c = 1/2$ give satisfactory stability results. For third order advection discretization, the Adams scheme allows somewhat larger Courant numbers. On the other hand, the BDF scheme has optimal damping properties for the implicit eigenvalues.

Remark 5.1. The bounds of Table 5.1 were determined experimentally (using Matlab graphics) and are sufficiently accurate for practical purposes. Upper bounds could be obtained by using the techniques of [11]. For the explicit two-step schemes it is even possible to determine maximal Courant numbers analytically by examining the characteristic polynomial. These examinations are elementary but the derivations involved are lengthy and readily become very cumbersome. In [10] a derivation is given for the explicit scheme in (2.11) and the third order upwind discretization. The final steps in this derivation have been carried out with Maple. To ten decimal digits accuracy, the maximal Courant number computed from this expression equals 0.4617485908.

We have carried out a similar derivation for the explicit Adams scheme (2nd order Adams-Bashforth) and the third order upwind discretization. In this case the maximal CFL number in ten decimal digits accuracy is equal to 0.5801977435. The maximum can be shown to be equal to

$$\min_{0 \le x \le 1} \nu(x),$$

where $\nu(x)$ is the real zero of the cubic equation

$$P_3(x)\,\nu^3 + P_2(x)\,\nu^2 - \frac{2}{3} = 0,$$

with

$$P_3(x) = \frac{(Q(x))^2}{162\,(x-1)^2}, \quad P_2(x) = \frac{1}{18}\,Q(x), \quad Q(x) = (x-1)\,(4x^2-5x-17).$$

It can be shown that the above cubic polynomial in ν has only one real root for $|x| \leq 1$, which means that $\nu(x)$ is defined by the well known formula of Cardano. However, the minimization over x is very complicated and at this stage Maple has to be used to find the (very long) analytical expression for the maximal Courant number given above.

References

[1] G. Akrivis, M. Crouzeix, and C. Makridadis. Implicit-explicit multistep finite element methods for nonlinear parabolic equations. Technical Report 95-22, University of Rennes, 1995.

[2] U.M. Ascher, S.J. Ruuth, and B. Wetton. Implicit-explicit methods for time-dependent PDE's. *SIAM J. Numer. Anal.*, 32:797–823, 1995.

[3] M. Crouzeix. Une méthode multipas implicite-explicite pour l' approximation des équations d' évolution paraboliques. *Numer. Math.*, 35:257–276, 1980.

[4] E. Hairer, S.P. Nørsett, and G. Wanner. *Solving Ordinary Differential Equations I – Nonstiff Problems.* Springer Verlag, Berlin, 1987.

[5] E. Hairer and G. Wanner. *Solving Ordinary Differential Equations II – Stiff and Differential-Algebraic Problems.* Springer Verlag, Berlin, 1991.

[6] W. Hundsdorfer and J.G. Verwer. A note on splitting errors for advection-reaction equations. *Appl. Num. Math.*, 18:191–199, 1995.

[7] J.D. Lambert. *Numerical Methods for Ordinary Differential Systems.* John Wiley & Sons, Chicester, 1991.

[8] O. Nevanlinna and W. Liniger. Contractive methods for stiff differential equations II. *BIT*, 19:53–72, 1979.

[9] J.M. Varah. Stability restrictions on second order, three-level finite-difference schemes for parabolic equations. *SIAM J. Numer. Anal.*, 17:300–309, 1980.

[10] J.G. Verwer, J.G. Blom, and W. Hundsdorfer. An implicit-explicit approach for atmospheric transport-chemistry problems. *Appl. Num. Math.*, 20:191–209, 1996.

[11] P. Wesseling. Von Neumann stability conditions for the convection-diffusion equation. *IMA J. Numer. Anal.*, 16:583–598, 1996.

[12] Z. Zlatev. *Computer Treatment of Large Air Pollution Models.* Kluwer Academic Publishers, Dordrecht, 1995.

Chapter 3

A parallel multiblock method for Poisson's equation

Abstract. Solution of large linear systems encountered in computational fluid dynamics often naturally leads to some form of domain decomposition, especially when it is desired to use parallel machines. It has been proposed to use approximate solvers to obtain fast but rough solutions on the separate subdomains. In this paper approximate solutions via (1) an inner preconditioned GMRES iteration to fixed tolerance and (2) incomplete factorization (RILU, restricted to the diagonal) are considered. Numerical experiments for a fundamental test problem are included which show speedups obtained on a cluster of workstations as well as on a distributed memory parallel computer. Additionally, the parallel implementation of GCR is addressed, with particular focus on communication costs associated with orthogonalization processes. This consideration brings up questions concerning the use of Householder reflections with GCR.

3.1 Introduction

Domain decomposition arises naturally in computational fluid dynamics applications on structured grids: complicated geometries are broken down into (topologically) rectangular regions and discretized in general coordinates, see e.g. [30, 40, 39], applying domain decomposition to iteratively arrive at the solution on the global domain. This approach provides easy exploitation of parallel computing resources, and additionally offers a solution to memory limitation problems.

This paper addresses the parallel implementation of a domain decomposition method for the DeFT Navier-Stokes solver described in [30], and is the continuation of work summarized in [7]. Results from a parallel implementation of a Krylov-accelerated Schur complement domain decomposition method are presented in [5]. A serial implementation of nonoverlapping, one-level additive Schwarz method with approximate subdomain solution [6] gave more promising results. For the present research, our goal was to obtain an impression of the behavior of this method in parallel without incurring the programming workload of a full implementation in the DeFT software; which would require fundamental changes. To this end, we report preliminary results for a Poisson problem on a square domain, and refer the reader to a forthcoming article with more realistic experiments. The Poisson problem is representative of the system which must be solved for the pressure correction method used in DeFT.

Theoretical results on approximate solution of subdomain problems for Schur complement domain decomposition methods are given by Börgers [4], Haase, Langer and Meyer [26, 17, 15, 16], and Cheng [9]. Brakkee [7] gives theoretical and experimental results for non-overlapping Schwarz iterations with variable approximate inner solvers.

In this paper we demonstrate that a reasonable amount of parallel speedup can be observed for a nonoverlapping, one-level additive Schwarz method if the subdomain problems are solved using only a rough approximation. In Section 3.2 we briefly review the relevant mathematics and give some theoretical motivation for approximate subdomain solution.

Much effort has focused on efficient parallelization of Krylov subspace methods. Aside from the preconditioning, the main parallel operations required in these methods are distributed matrix-vector multiplications and inner products. For many problems, the matrix-vector multiplications require only nearest neighbor communications, and may be very efficient. Inner products, on the other hand, require global communications; therefore, the focus has been on reducing the number of inner products [12, 31], overlapping inner product communications with computation [10], or increasing the number of inner products that can be computed with a single communication [2, 24].

Some practical points are brought out in Section 3.3 concerning parallel implementation of orthogonalization procedures for the GCR method. A performance model is developed for comparison of these methods, and the validity of the model is checked against experimental results in Section 3.4.

Additional results reported in Section 3.4 include speedup ratios, obtained by comparison of the parallel multiblock computation times to both the single block serial time and the multiblock serial times, and scalability tests for which the number of unknowns per processor is held constant as the number of participating processors is increased. The timings were made on a cluster of workstations and a Cray T3E. In particular, our results suggest that the most efficient subdomain approximation in terms of computation time is a simple incomplete factorization.

3.2 Mathematical Background

3.2.1 One-level, nonoverlapping domain decomposition

We consider an elliptic partial differential equation discretized using a finite volume or finite difference method on a computational domain Ω. By a computational domain we mean the set of unknown values to be approximated, together with their associated locations in space. Let the domain be the union of M nonoverlapping subdomains Ω_m, $m = 1, \ldots, M$.

Discretization of the PDE results in a sparse linear system

$$Ax = b, \tag{3.1}$$

with x, $b \in \mathbb{R}^N$. The structure of the matrix A is determined by the stencil of the discretization. Even if there is no overlap between the subdomains, there is an inter-subdomain coupling due to the stencil. That is, the equation for an unknown adjacent to a subdomain interface is dependent on an unknown across the subdomain boundary.

One technique for solving this problem is to permute the system (3.1), grouping together into blocks those unknowns which share a common subdomain to produce a block system:

$$\begin{bmatrix} A_{11} & \dots & A_{1M} \\ \vdots & \ddots & \vdots \\ A_{M1} & \dots & A_{MM} \end{bmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_M \end{pmatrix} = \begin{pmatrix} b_1 \\ \vdots \\ b_M \end{pmatrix}. \tag{3.2}$$

In this system, one observes that the diagonal blocks A_{mm} express coupling among the unknowns defined on a common subdomain (Ω_m), whereas the off-diagonal blocks A_{mn}, $m \neq n$ represent coupling across subdomain boundaries. The only nonzero off-diagonal blocks are those corresponding to neighboring subdomains.

The additive Schwarz iteration introduces the block Jacobi preconditioner:

$$K = \begin{bmatrix} A_{11} & & \\ & \ddots & \\ & & A_{MM} \end{bmatrix},$$

which, together with the residual, defines a system whose solution provides an approximation of the error. Note that this system may be efficiently solved on parallel computers. It is this form of domain decomposition which we will consider in the rest of the paper.

For a thorough discussion of domain decomposition methods see the book [32] and the review article [8]. Each of these publications contains an extensive bibliography. Convergence theory for domain decomposition methods is discussed in [32]. Roughly speaking, the convergence rate suffers proportionally to the number of subdomains in each direction. If a constant overlap (in physical units) is maintained, the convergence rate is independent of grid size; however, for zero overlap the convergence is relatively poor. The convergence rate may additionally be made independent of the number of subdomains if a coarse subspace correction is applied: for example, the residual is projected onto a single coarse grid domain, where a correction is computed which is then interpolated back to the subdomains.

3.2.2 Krylov subspace acceleration

In practice (3.2) is solved iteratively, using K as a preconditioner for a Krylov subspace method, such as the conjugate gradient method for symmetric problems or the GMRES method [29] for nonsymmetric problems. For our purposes a practical method is GCR

```
Algorithm: GCR
Given: initial guess $x_0$
$r_0 = b - Ax_0$
for $k = 1, \dots,$ convergence
  Solve $K\tilde{v} = r_{k-1}$ (approximately)
  $\tilde{q} = A\tilde{v}$
  $[q_k, v_k] =$ orthonorm $(\tilde{q}, \tilde{v}, q_i, v_i, i < k)$
  $\gamma = q_k^T r_{k-1}$
  Update: $x_k = x_{k-1} + \gamma v_k$
  Update: $r_k = r_{k-1} - \gamma q_k$
end
```

FIGURE 3.1: The GCR Algorithm

[11], shown in Figure 3.1. In the algorithm and elsewhere in this paper the Euclidean inner product $\langle x, y\rangle = x^T y$ and associated norm $\|x\| = (x^T x)^{1/2}$ are used.

The function **orthonorm()** takes input vectors $\tilde{q}$ and $\tilde{v}$, orthonormalizes $\tilde{q}$ with respect to the q_i, $i < k$, updating $\tilde{v}$ as necessary to preserve the relation $\tilde{q} = A\tilde{v}$, and returns the modified vectors q_k and v_k. In serial computations, the modified Gram-Schmidt method, Figure 3.2, is often employed for the **orthonorm()** function. We discuss alternative orthogonalization methods in later sections of this paper.

```
Algorithm: Modified Gram-Schmidt
$[q_k, v_k] =$ orthonorm $(\tilde{q}, \tilde{v}, q_i, v_i, i < k)$:
  for $i = 1, \dots, k-1$
    $\alpha = \langle \tilde{q}, q_i \rangle$
    $\tilde{q} = \tilde{q} - \alpha q_i$
    $\tilde{v} = \tilde{v} - \alpha v_i$
  end
  $\beta = \|\tilde{q}\|$
  $q_k = \tilde{q}/\beta$; $v_k = \tilde{v}/\beta$
return
```

FIGURE 3.2: The modified Gram-Schmidt algorithm

In exact arithmetic, and assuming it does not break down, GCR produces the same iterates as GMRES. However, GCR does not take advantage of the Lanczos recursion, but instead requires the storage of an extra set of orthogonal *residual search vectors*. GCR has a number of benefits; among them: (1) the preconditioner K need not remain constant (nor even be a linear operator; the GMRESR algorithm in [34] uses GMRES(m) as a preconditioner); (2) one is free to employ truncation strategies such as in [35]; and (3) if the LSQR switch is employed [34], the method will not break down. The importance of allowing a variable preconditioner will be discussed in the next section. In Figure 3.1 the GCR method is defined for an unlimited number of iterations, and may incur memory limitations. In practice, therefore, it is necessary either to restart the iteration periodically, discarding all stored vectors, or to maintain only a fixed number of vectors, applying some criterion to determine which vectors will be kept. This second option, referred to as truncation, is shown in [35] to be very effective in reducing the number of iterations. In the numerical experiments of this paper we do not use any truncation strategies, but simply restart; however, truncation is used in the DeFT software.

3.2.3 Approximate subdomain solution

Solution for $\tilde{v}$ from the preconditioning equation $K\tilde{v} = r_{k-1}$ in the GCR algorithm requires solution of M subdomain systems $A_{mm}\tilde{v}_m = r_m$, $m = 1, \ldots, M$. Since these problems have a nonzero structure similar to that of the original matrix A, and since they may still be quite large, it is advantageous to solve them using an iterative method. A question which arises naturally, and for purely practical reasons, addresses the tolerance to which these *inner iterations* should converge. Perhaps a very rough approximation would be sufficient. A number of authors have considered approximate solution of subdomain problems. In particular they have considered the consequences of using very fast, rough approximations to reduce the total computing time necessary to solve the global problem.

Some possible strategies for approximating the subdomain solutions are:

- A second (inner) iterative method (possibly preconditioned) either to a fixed tolerance, to a variable tolerance, or for a predetermined number of iterations.
- Approximate factorization or approximate inversion of the subdomain problems.
- Do nothing at all. In this case one uses the domain decomposition purely as a form of data distribution and applies the unpreconditioned Krylov method.

Tan [33] shows that if the inner problems are solved to some tolerance in each outer iteration, then the optimal strategy for choosing the tolerance is a fixed one. That is, it is not necessary to make the subdomain solution tolerance smaller as the global solution converges.

Brakkee [7] has proven the following theorem. Let $\tilde{A}_{ii}^{-1}$ be the matrix which represents the approximate inversion of the ith block. In the case of a Krylov subspace method as inner solver, this would be the actual value of the minimizing polynomial applied to A. Similarly define $\tilde{K}^{-1}$ to be the approximate preconditioner consisting of the diagonal blocks $\tilde{A}_{ii}^{-1}$. If for each subdomain $i = 1, \ldots, M$ it holds that $\|I - A_{ii}\tilde{A}_{ii}^{-1}\| < \epsilon$, then the condition number of the approximately preconditioned matrix satisfies

$$\kappa(A\tilde{K}^{-1}) \leq \frac{1+\epsilon}{1-\epsilon}\kappa(AK^{-1}). \tag{3.3}$$

where $\kappa(A) = \|A\|\|A^{-1}\|$ is the condition number of A. Unfortunately, the condition $\|I - A_{ii}\tilde{A}_{ii}^{-1}\| < \epsilon$ is nontrivial to check.

Essential to the proof of the above theorem is the fact that

$$\kappa(A\tilde{K}^{-1}) = \kappa(AK^{-1}K\tilde{K}^{-1}) \leq \kappa(AK^{-1})\kappa(K\tilde{K}^{-1}).$$

This bound may be clarified by noting that the matrix $B = K\tilde{K}^{-1}$ is a block diagonal matrix with blocks $B_i = A_{ii}\tilde{A}_{ii}^{-1}$, $i = 1, \ldots, M$. The spectrum of the block diagonal matrix B^TB is a subset of the union of the spectra of the blocks; thus, if there exist α, β bounding the singular values of all blocks: $0 < \alpha \leq \min_i \sigma_{\min}(B_i) \leq \max_i \sigma_{\max}(B_i) \leq \beta$, then

$$\kappa(A\tilde{K}^{-1}) \leq \frac{\beta}{\alpha}\kappa(AK^{-1}).$$

Competing against convergence rate for an efficient solution method is the expense of computing the subdomain approximate solutions. The RILUD preconditioner, though a less effective approximate solver than GMRES iterations in terms of convergence rate, is far cheaper, at least for the problems considered here. Thus one makes a tradeoff between effectiveness of an approximate preconditioner in terms of convergence rate and speed in terms of computational expense.

Note that if the subdomains are solved using a Krylov subspace method such as GMRES, then the approximate solution is a function of the right hand side, which is the residual of the outer iteration. Furthermore, if the subdomains are solved to a tolerance, the number of inner iterations may vary from one subdomain to another, and in each outer iteration. The effective preconditioner is therefore nonlinear and varies in each outer iteration. A variable preconditioner presents a problem for GMRES: namely, the Lanczos recurrence relation no longer holds. To allow the use of a variable preconditioner, Saad [28] has developed the Flexible GMRES (FGMRES) method, which requires storage of an auxiliary set of vectors such as with GCR. However, it is not possible to use truncation strategies with FGMRES. Because we use truncation in our Navier-Stokes code, we consider GCR in the following.

Our choice of approximate solution methods is motivated by the results obtained in [6]. In that paper, GMRES was used as to approximately solve subdomain problems to within fixed tolerances of 10^{-4}, 10^{-3}, 10^{-2} and 10^{-1}. Additionally, a blockwise application of the RILUD preconditioner was used. RILUD, a diagonal-restricted variant of the preconditioner introduced in [1], is a weighted average of an ILUD preconditioner [25] and an MILUD preconditioner [14]. The weighting parameter ω, was assigned a value of 0.95 in our experiments. See also [37] for useful results with RILU factorizations applied to Navier-Stokes equations. The use of incomplete factorizations to obtain subdomain approximations has been advocated by Keyes [22] and Goossens et al. [13] among others. The results of [6] indicated that coarser tolerances were more effective. However, all numerical results presented therein were obtained from serial runs. In Section 3.4 we will present numerical results using the above approximate subdomain solution methods in parallel.

3.2.4 Orthogonalization methods

The primary challenges to parallelization of GCR are parallelization of the preconditioning—a difficulty which disappears when a block preconditioner K is used—and parallel computation of the inner products. Inner products require global communication and therefore do not scale. Much of the literature on parallel Krylov subspace methods and parallel orthogonalization methods is focused on orthogonalizing a number of vectors simultaneously. See, e.g. [27, 20, 2, 10, 24]. However, this is not possible using a preconditioner which varies in each iteration. For this reason, we need a method for orthogonalizing one new vector against an orthonormal basis of vectors.

The modified Gram-Schmidt method of Figure 3.2 suffers from the fact that the inner products must be computed using successive communications, and the number of these inner products increases by one with the iteration number. This is not the case if one uses the classical Gram-Schmidt method, Figure 3.3. In this algorithm all necessary inner products can be computed with a single global communication. Unfortunately, as shown by Björck [3], the classical Gram-Schmidt method is unstable with respect to rounding errors, so this method is rarely used.

```
Algorithm: Classical Gram-Schmidt
$[q_k, v_k] =$ orthonorm $(\tilde{q}, \tilde{v}, q_i, v_i, i < k)$:
  $\beta = \langle \tilde{q}, \tilde{q} \rangle$
  for $i = 1, \ldots, k-1$
    $\alpha_i = \langle \tilde{q}, q_i \rangle$
  end
  $\beta = \sqrt{\beta - \sum_{i=1}^{k-1} \alpha_i^2}$
  $q_k = \beta^{-1} \left( \tilde{q} - \sum_{i=1}^{k-1} \alpha_i q_i \right)$
  $v_k = \beta^{-1} \left( \tilde{v} - \sum_{i=1}^{k-1} \alpha_i v_i \right)$
return
```

FIGURE 3.3: The classical Gram-Schmidt algorithm

On the other hand, Hoffmann [19] gives experimental evidence indicating that a two-fold application of Figure 3.3 *is* stable. Furthermore, it appears that if orthogonality is important, such a re-orthogonalization is also required even for the more stable modified Gram-Schmidt algorithm.

A third method which has been suggested is the parallel implementation of Householder transformations, introduced by Walker [38]. We shall reformulate that method for GCR in the following section. Additionally, we will present a simple parallel performance analysis for comparison of these three orthogonalization procedures.

3.3 Householder orthogonalization

Walker [38] has proposed a GMRES variant using a vectorized version of Householder transformations as an alternative to the modified Gram-Schmidt procedure. The Householder method has the advantage that it requires only a fixed number of communications per GMRES iteration. In this section we describe the GCR implementation and discuss some practical details concerning its use.

3.3.1 Description of the method

In the following discussion we use the notion a_k to represent the kth column of a matrix A and $a^{(i)}$ to represent the ith component of a vector a. Let a matrix $A \in \mathbb{R}^{n\times m}$, $m \leq n$ with linearly independent columns be factored as QZ, where Q is orthogonal and Z is upper triangular. Then the kth column of A is given by $a_k = Qz_k$. It follows that $a_k \in \text{span}\{q_1, \ldots, q_k\}$. In other words, the columns of Q form an orthonormal basis for the span of the columns of A.

We construct Q as the product of a series of Householder reflections, $Q = P_1 \cdots P_m$, used to transform A into Z. The matrices P_i have the following properties:

i) $P_i^2 = I = P_i^T P_i$.

ii) $P_i e_j = e_j$, if $j < i$.

iii) $P_i(P_{i-1} \cdots P_1)a_i = z_i$.

In property ii) e_j is the jth canonical unit vector in $\mathbb{R}^n$. A Householder reflection is given by $P_i = I - 2\frac{w_i w_i^T}{w_i^T w_i}$, for some $w_i \in \mathbb{R}^n$. Note that such a matrix has property i). Property ii) is ensured by requiring the first $i-1$ components of w_i be zero: $w_i^{(j)} = 0$ for $j < i$.

Suppose one has already produced k orthogonal basis vectors $q_1, \ldots, q_k$ and stored them along with the transformation vectors $w_1, \ldots, w_k$ corresponding to $P_1, \ldots, P_k$. Given a candidate vector a_{k+1}, one must first apply the previous reflections as described in [38]:

$$\tilde{a} = P_k \cdots P_1 a_{k+1} = (I - 2W_k L_k^{-1} W_k^T) a_{k+1}$$

where here and elsewhere we denote by W_k the matrix whose columns are $w_1, \ldots, w_k$, and where

$$L_k = \begin{bmatrix} 1 & & & \\ 2w_2^T w_1 & 1 & & \\ \vdots & & \ddots & \\ 2w_k^T w_1 & \ldots & 2w_k^T w_{k-1} & 1 \end{bmatrix}.$$

Note especially that in the $(k+1)$th iteration one must compute the last row of L_k, which is the vector $(2w_k^T W_{k-1}, 1)$, as well as the vector $W_k^T a_{k+1}$. This requires $2k-1$ inner products, but they may all be computed using only a single global communication.

Now having computed $\tilde{a}$ one wishes to find w_{k+1} such that P_{k+1} satisfies iii):

$$\begin{aligned} P_{k+1}\tilde{a} = z_{k+1} &= z_{k+1}^{(1)} e_1 + \cdots + z_{k+1}^{(k+1)} e_{k+1} \\ &= \tilde{a}^{(1)} e_1 + \cdots + \tilde{a}^{(k)} e_k + \alpha e_{k+1}, \end{aligned} \tag{3.4}$$

where property ii) has been used for the last equality.

Because of the relation

$$P_{k+1}\tilde{a} = (I - 2\frac{w_{k+1}w_{k+1}^T}{w_{k+1}^T w_{k+1}})\tilde{a} = \tilde{a} - 2\frac{w_{k+1}^T \tilde{a}}{w_{k+1}^T w_{k+1}} w_{k+1}, \tag{3.5}$$

one must have $w_{k+1} \in \operatorname{span}\{\tilde{a}, e_1, \ldots, e_{k+1}\}$. However equation (3.4) provides the relation which must hold among $\tilde{a}, e_1, \ldots, e_k$. Let $\tilde{w}$ be the vector obtained by setting the first k elements of $\tilde{a}$ to zero. Formally, one has $\tilde{w} = J_{k+1}\tilde{a}$, where

$$J_{k+1} = \begin{bmatrix} 0_k & \\ & I_{n-k} \end{bmatrix}.$$

Thus, $w_{k+1} \in \operatorname{span}\{\tilde{w}, e_{k+1}\}$. The length of w_{k+1} is a free parameter, so take $w_{k+1} = \tilde{w} + \beta e_{k+1}$. Substituting into (3.5) gives

$$\begin{aligned} P_{k+1}\tilde{a} &= \tilde{a} - 2\frac{w_{k+1}^T \tilde{a}}{w_{k+1}^T w_{k+1}}(J_{k+1}\tilde{a} + \beta e_{k+1}) \\ &= (I - 2\frac{w_{k+1}^T \tilde{a}}{w_{k+1}^T w_{k+1}} J_{k+1})\tilde{a} - 2\beta\frac{w_{k+1}^T \tilde{a}}{w_{k+1}^T w_{k+1}} e_{k+1}. \end{aligned}$$

To ensure that all elements below the $(k+1)$th are zero, one requires $1 - 2\frac{w_{k+1}^T \tilde{a}}{w_{k+1}^T w_{k+1}} = 0$. But,

$$w_{k+1}^T \tilde{a} = (\tilde{w} + \beta e_{k+1})^T \tilde{a} = \|\tilde{w}\|^2 + \beta \tilde{a}^{(k+1)},$$

and

$$w_{k+1}^T w_{k+1} = \|(\tilde{w} + \beta e_{k+1})\|^2 = \|\tilde{w}\|^2 + 2\beta\tilde{a}^{(k+1)} + \beta^2.$$

Substituting these numbers into the above relation, one finds $\beta = \pm\|\tilde{w}\|$, and the sign of β is chosen to be the same as that of $\tilde{w}^{(k+1)}$ to reduce the risk of subtractive cancellation:

$$w_{k+1} = \tilde{w} + \text{sign}\,(\tilde{w}^{(k+1)})\|\tilde{w}\|e_{k+1}.$$

In practice, the w_k are normalized to length one. Since α is the $(k+1)$th component of $P_{k+1}\tilde{a}$, substitution of the above relation into (3.5), and noting that

$$2\frac{w_{k+1}^T\tilde{a}}{w_{k+1}^T w_{k+1}} = 1,$$

gives $\alpha = \tilde{a}^{(k+1)} - \tilde{w}^{(k+1)} - \text{sign}\,(\tilde{w}^{(k+1)})\|\tilde{w}\| = -\text{sign}\,(\tilde{w}^{(k+1)})\|\tilde{w}\| = -\beta$, and the length of w_{k+1} can be expressed as

$$\|w_{k+1}\| = \sqrt{2\alpha^2 - 2\alpha\tilde{w}^{(k+1)}}.$$

The $(k+1)$th column of Q is the new orthonormal basis vector,

$$Qe_{k+1} = P_1 \cdots P_{k+1}e_{k+1},$$

and because of property ii), the yet to be computed reflections will not affect this column. Multiplying both sides of (3.4) by $P_1 \cdots P_{k+1}$ gives:

$$a_{k+1} = \tilde{a}^{(1)}q_1 + \cdots + \tilde{a}^{(k)}q_k + \alpha q_{k+1},$$

from which it follows that

$$q_{k+1} = \frac{1}{\alpha}\left[a_{k+1} - \sum_{i=1}^{k}\tilde{a}^{(i)}q_i\right]. \tag{3.6}$$

Within the GCR algorithm, the same linear combination must be applied to the v_i to obtain v_{k+1}. As pointed out by an anonymous referee, use of equation (3.6) may result in subtractive cancellation, causing the reduced stability of this Householder implementation observed in the next section. However, a relation of this form is needed to be able to enforce the identity $q_{k+1} = Av_{k+1}$, necessary for GCR.

Our implementation requires three communications in the $k+1$th iteration, namely:

1. The computation of $\tilde{a}$ using Walker's approach, requires $2k-1$ inner products, all of which can be performed with a single communication.
2. A second communication is necessary to broadcast the first $k+1$ elements of $\tilde{a}$.
3. A communication is required to compute $\|\tilde{w}\|$ for determining α.

Algorithm: Householder Orthogonalization

```
$[q_k, v_k]$ = orthonorm $(\tilde{q}, \tilde{v}, q_i, v_i, i < k)$:
  if $k == 1$
    $w_1 = \tilde{q}$
  else
    if $k == 2$
      $L_1 = 1$
    else
      $L_{k-1} = \begin{bmatrix} L_{k-2} & 0 \\ 2w_{k-1}^T W_{k-2} & 1 \end{bmatrix}$
    end
    $y = W_{k-1}^T \tilde{q}$
    Solve $L_{k-1} d = y$
    $w_k = \tilde{q} - 2W_{k-1} d$
  end
  $\tilde{a}^{(i)} = w_k^{(i)}, i = 1, \ldots, k$
  Broadcast $(\tilde{a})$
  $q_k = \tilde{q} - \sum_{i<k} \tilde{a}^{(i)} q_i$
  $v_k = \tilde{v} - \sum_{i<k} \tilde{a}^{(i)} v_i$
  Set $w_k^{(i)} = 0, i = 1, \ldots, k-1$
  $\alpha = -\mathrm{sign}(\tilde{a}^{(k)}) \|w_k\|$
  $w_k^{(k)} = w_k^{(k)} - \alpha$
  $q_k = q_k / \alpha$
  $v_k = v_k / \alpha$
  $w_k = w_k / \sqrt{2\alpha(\alpha - \tilde{a}^{(k)})}$
return
```

FIGURE 3.4: The Householder orthogonalization algorithm

The algorithm is shown in Figure 3.4, with $\tilde{q}$ playing the role of a in the above discussion.

Comparing the Householder implementation with modified Gram-Schmidt,

- In the kth iteration the Householder method requires three communications, whereas Gram-Schmidt requires $k + 1$.
- Householder requires approximately twice as many inner products as Gram-Schmidt, plus $1\frac{1}{2}$ times the number of 'axpy' operations.
- The Householder method requires the storage of an extra set of k vectors.

A drawback of the Householder method is that there appears to be no simple way to incorporate truncation schemes in the GCR method if Householder is used for orthogonalization.

In the next section we develop a performance model for comparison of the Householder and Gram-Schmidt methods.

3.3.2 Performance model

To give insight into the choice of an orthogonalization procedure, consider a simple performance model. Let the time required for communication of a message of n floating point numbers be given by

$$t_{\text{comm}} = t_0 + \beta n,$$

where t_0 is the fixed time required for a message of length zero, and β is the time per floating point number (bandwidth). Let the time for n floating point operations be given by

$$t_{\text{comp}} = \phi n,$$

where ϕ is the time for 1 floating point operation. Similar computation/communication models are used, for example, in [10, 27, 18, 23].

Let p denote the number of processes, and define a function $f(p)$ which gives the maximum number of non-simultaneous sends necessary for any given process participating in a broadcast operation to $p-1$ processes. The function $f(p)$ is machine-dependent and also dependent on the distribution of processes on the machine. For example, assuming perfect connectivity and that processes not participating in a given communication are free to participate in a concurrent communication, the broadcast of a message among p processes requires $f(p) = \lceil \log_2 p \rceil$ consecutive send operations from the broadcasting process. An Ethernet broadcast requires $f(p) = p-1$ consecutive send operations.

Assume each processor is responsible for an $n \times n$ subdomain with n^2 unknowns. Define the times for some basic operations:

Op.	Communication	Computation	Definition
send (k)	$t_0 + \beta k$		send a message of length k
flop (n)		$n\phi$	n floating point operations
$B(p,k)$	$f(p)(t_0 + \beta k)$		broadcast k elements
$G(p,k)$	$2f(p)(t_0 + \beta k)$		global sum k elements
SIP(k)	$G(p,k)$	$2kn^2\phi$	k inner prod. simult. comms.
FS(k)		$k^2\phi$	forward substitution, order k
axpy		$2n^2\phi$	$z = ax + y$, scalar a

Note that we distinguish between inner products that can be computed simultaneously (i.e. with a single communication) and inner products that cannot. For example, k simultaneous inner products are denoted SIP(k), whereas k non-simultaneous inner products are denoted k SIP(1). The modified and re-orthogonalized classical Gram-Schmidt and Householder routines can be broken down into components as follows. In the kth iteration of GCR:

Mod. Gram-Schmidt	k SIP(1) $2k-1$ axpy
Re-orthog. CGS	2 SIP (k) $3k-1$ axpy
Householder	SIP ($2k-3$) SIP(1) FS ($k-1$) $3k-3/2$ axpy 1 B (p,k)

We have implemented the re-orthogonalized classical Gram-Schmidt method so that in the kth iteration, the candidate residual search vector $\tilde{q}$ is twice orthogonalized against the basis $q_1, \ldots, q_{k-1}$ to obtain q_k, and only then is the search vector v_k computed. This eliminates a series of vector updates and explains why there are only $3k-1$ 'axpy' operations.

Based on the communication model outlined in the previous two tables, the orthogonalization time required for s iterations of GCR (without restart) using the modified Gram-Schmidt (MGS), re-orthogonalized classical Gram-Schmidt (CGS2) and Householder (HH) methods, respectively, is given by:

$$t_{\mathrm{MGS}} = \frac{s(s+1)}{2}\left[6n^2\phi + 2f(p)(t_0+\beta)\right] - s(2n^2\phi), \tag{3.7}$$

$$t_{\mathrm{CGS2}} = \frac{s(s+1)}{2}\left[10n^2\phi + 4f(p)\beta\right] + s\left[4f(p)t_0 - 2n^2\phi\right], \tag{3.8}$$

$$\begin{aligned} t_{\mathrm{HH}} =& \phi\frac{s(s+\frac{1}{2})(s+1)}{3} + \frac{s(s+1)}{2}\left[(10n^2-2)\phi + 5f(p)\beta\right] \\ &+ s\left[f(p)(6t_0-4\beta) - (7n^2-1)\phi\right]. \end{aligned} \tag{3.9}$$

If the forward substitution in the Householder algorithm is negligible, the model becomes

$$t_{\mathrm{HH}} = \frac{s(s+1)}{2}\left[10n^2\phi + 5f(p)\beta\right] + s\left[f(p)(6t_0-4\beta) - 7n^2\phi\right].$$

Comparing this expression with (3.8) for the re-orthogonalized classical Gram-Schmidt algorithm, we see that the two methods are very similar in cost, while we shall see later that re-orthogonalized Gram-Schmidt is much more stable than Householder for the standard test problem. The similarity in cost is confirmed by our experiments.

Tests were performed on a cluster of HP workstations to obtain representative values for the parameters t_0, β and ϕ:

$$t_0 \approx 4.7 \times 10^{-4}, \qquad \beta \approx 7.5 \times 10^{-6}, \qquad \phi \approx 4.9 \times 10^{-8}.$$

Similar tests were performed on a Cray T3E using MPI communications with the results:

$$t_0 \approx 2.4 \times 10^{-5} \qquad \beta \approx 5.4 \times 10^{-8} \qquad \phi \approx 5.8 \times 10^{-8}.$$

Assuming the models (3.7), (3.8)and (3.9), and assuming $f(p) = p - 1$ for the workstation cluster and $f(p) = \lceil \log_2 p \rceil$ for the Cray T3E, the quantities

$$\mathcal{F}_{\text{HH}} = \frac{\text{orthog. time MGS}}{\text{orthog. time HH}}, \tag{3.10}$$

$$\mathcal{F}_{\text{CGS2}} = \frac{\text{orthog. time MGS}}{\text{orthog. time CGS2}} \tag{3.11}$$

are plotted as a function of n for $s = 60$ and $p = 4, 9$ ($p = 4, 9, 25$ for the Cray T3E) in Figure 3.5. The Householder (resp. CGS2) method is faster at points in the figure where $\mathcal{F}_{\text{HH}} > 1$ (resp. $\mathcal{F}_{\text{CGS2}} > 1$). The model predicts that the alternative methods (HH) and (CGS2) are only advantageous for small enough subdomain size. On the workstation cluster this size may be about 10000 unknowns on 4 processors and somewhat more on 9 processors. On the Cray T3E, the number of unknowns per processor should be fewer than 1000 for 9 or even 25 processors. For larger problems the smaller amount of work involved in modified Gram-Schmidt orthogonalization outweighs the increased communication cost. Note also that the model indicates that the computational efforts of Householder and re-orthogonalized classical Gram-Schmidt are very similar, with the Gram-Schmidt variant to be preferred in the useful range.

In the following section we shall see that, while the model is qualitatively correct, the observed performance curves are lower than the ones predicted here.

Another issue of relevance to the choice of an orthogonalization method is the stability of the method with respect to rounding errors. Figure 3.6 shows a comparison of the classical, modified, and re-orthogonalized classical Gram-Schmidt methods and the Householder implementation for the test matrix of [3]:

$$A = \begin{bmatrix} 1 & 1 & \dots \\ \epsilon & & \\ & \epsilon & \\ & & \ddots \\ & & \end{bmatrix}.$$

The comparison method is the QR decomposition function of Matlab, based on the traditional Householder implementation. We see that the re-orthogonalized classical

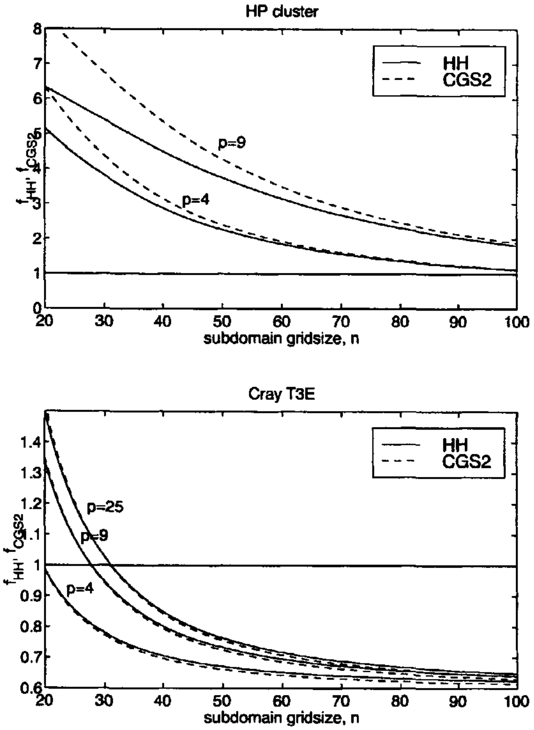

FIGURE 3.5: Predicted speedup with Householder orthogonalization

Gram-Schmidt method gives the smallest orthogonalization error of all methods for this test case.

In conclusion, we mention that there does not seem to be any reason to prefer the parallel Householder method over the re-orthogonalized classical Gram-Schmidt method, at least in this context. In terms of parallel efficiency the two methods are almost identical. However the Gram-Schmidt variant is simpler to implement, provides significantly less orthogonalization error, and allows truncation strategies to be employed in a natural way.

3.4 Numerical experiments

In this section, we give numerical results which provide useful insights into approximate solution techniques. Numerical results were obtained from both a cluster of HP-735 and HP-755 workstations (99-125 MHz) and from a Cray T3E parallel computer. All

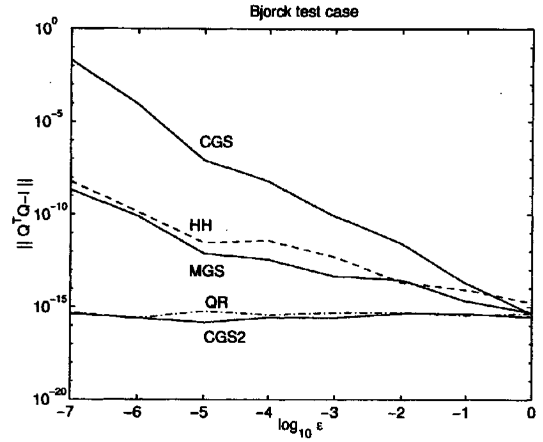

FIGURE 3.6: Comparison of orthogonalization error for classical (CGS), modified (MGS), reorthogonalized (CGS2) Gram-Schmidt methods, Householder (HH) method, and Matlab QR function on Björck test problem.

communications were handled with MPI. Reported times are obtained from the MPI timing functions, and are the minimum time achieved over three runs. For our interests, the workstation results are as important (or more so) than those from the parallel machine, due to the immediate availability and relative cheapness of workstations.

As a test example, we consider a Poisson problem, discretized with the finite volume method on a square domain. The pressure correction matrix, which we solve in each time step of an incompressible Navier-Stokes simulation to enforce the divergence-free constraint [21], is similar to a Poisson problem, but with asymmetry arising from the use of curvilinear coordinates. Solution of this system requires about 75% of the computing effort. So that we can obtain a useful indication of the performance of our method on the pressure correction matrix, we do not exploit the symmetry of the Poisson matrix in these experiments. The domain is composed of an $M \times M$ array of subdomains, each with an $n \times n$ grid. With $h = \Delta x = \Delta y = 1.0/(Mn)$ the discretization is

$$4u_{ij} - u_{i+1j} - u_{i-1j} - u_{ij-1} - u_{ij+1} = h^2 f_{ij}.$$

The right hand side function is $f_{ij} = f(ih, jh)$, where $f(x, y) = -32(x(1-x)+y(1-y))$. Homogeneous Dirichlet boundary conditions $u = 0$ are defined on $\partial\Omega$, implemented by adding a row of ghost cells around the domain, and enforcing the condition, for example, $u_{0j} = -u_{1j}$ on boundaries. This ghost cell scheme allows natural implementation of the domain decomposition as well.

3.4.1 Evaluation of performance model, Householder orthogonalization

The performance model for the orthogonalization methods in the previous section predicts that the modified Gram-Schmidt algorithm is to be preferred for large subdomain problems. We wish to investigate this experimentally, to confirm the model predictions. The results presented here were computed for a fixed number of iterations s, equal to the restart value.

Figure 3.7 is the experimental analog of Figure 3.5. The parameters $\mathcal{F}_{\text{HH}}$ and $\mathcal{F}_{\text{CGS2}}$ are plotted for subdomain grid sizes of $n = 20, 40, 60, 80, 100$ and a fixed number of iterations $s = 60$. Measurements were made for 4 and 9 processors ($M = 2, 3$, respectively) on the HP cluster and 4, 9 and 25 processors ($M = 2, 3, 5$, respectively) on the Cray T3E.

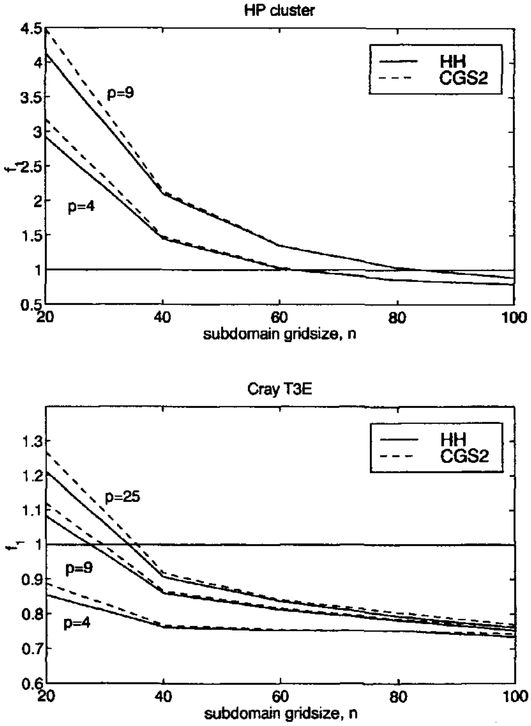

FIGURE 3.7: Measured speedup with Householder (HH) orthogonalization and re-orthogonalized classical Gram-Schmidt (CGS2), restart value $s = 60$.

By comparison one sees that the model developed in the previous section is qualitatively correct, but is rather optimistic with respect to the range of problem sizes for which

Householder is more effective than Gram-Schmidt.

3.4.2 Evaluation of approximate subdomain solvers

In this section we compare speedups obtained with a number of approximate subdomain solvers to get an impression of which solvers might be effectively used with the Navier-Stokes equations. For the tests of this section, a fixed restart value of $s = 30$ was used, and modified Gram-Schmidt was used as the orthogonalization method for all computations. The solution was computed to a fixed tolerance of 10^{-6} unless noted otherwise. The performance measure is computation time, after initialization, taken as the minimum achieved over three runs.

The subdomain approximations will be denoted as follows:

- GMR6 = restarted GMRES with a tolerance of 10^{-6}, (preconditioned with RILUD)
- GMR2 = restarted GMRES with a tolerance of 10^{-2}, (preconditioned with RILUD)
- GMR1 = restarted GMRES with a tolerance of 10^{-1}, (preconditioned with RILUD)
- RILUD = one application of an RILUD preconditioner.

Speedups are compared both to single and multiblock serial computations.

Single block serial case

The single block serial solution times in seconds on grids of dimension n = 60, 120, 180, 240 and 300 are, on the HP cluster:

	$n = 60$	120	180	240	300
GMR6	0.788	7.56	28.1	82.5	195
GMR2	0.862	8.00	34.7	75.8	180
GMR1	0.815	6.75	29.3	82.1	166
RILUD	1.10	11.0	41.6	117	292

and on the Cray T3E:

	$n = 60$	120	180	240	300
GMR6	0.483	3.98	11.9	34.7	80.6
GMR2	0.563	4.24	14.8	32.0	74.9
GMR1	0.552	3.62	13.2	35.4	69.3
RILUD	0.666	5.49	17.2	49.9	119

Note that GCR preconditioned with GMRES iterations gives a variation of the GMRESR method of [34]. All three lead to approximately the same solution time. This is in agreement with the findings of [34, 35, 36] for the GMRESR method. The fourth case is equivalent to solving the problem with GCR, preconditioned with the RILUD preconditioner. It is also in keeping with the findings of the above papers that this method is slower than GMRESR.

Multiblock solution, fixed problem size

In this section we compare results for a fixed problem size on the 300 × 300 grid with 4 and 9 processors on the workstation cluster and 4, 9, 16 and 25 processors on the Cray T3E. We use one processor per block. The timing results in seconds are, for the HP cluster:

	$p=4$	$p=9$
GMR6	1430	386
GMR2	346	220
GMR1	457	261
RILUD	157	89

and for the Cray T3E:

	$p=4$	$p=9$	$p=16$	$p=25$
GMR6	685	178	143	79.3
GMR2	167	102	63.3	37.1
GMR1	222	118	65.6	38.9
RILUD	65.3	25.9	21.9	14.9

On both systems one observes that the method using RILUD as the subdomain approximation gives a faster computation time than the fastest serial computation times from the previous subsection. On the Cray T3E, the methods GMR1 and GMR2 are also somewhat faster than the fastest serial time, for $p = 16$ and $p = 25$ processors.

Furthermore, one sees that among those methods in which GMRES is used to solve the subdomain problems, a tolerance of 10^{-2} gives a faster solution time than a tolerance of 10^{-1}. Thus some subdomain convergence appears to be desirable. On the other hand, the fastest solutions in each case are obtained with the least accurate subdomain approximation—namely, the RILUD preconditioner.

To give insight into these results, it is useful to look at the iteration counts: both the number of outer iterations and the average number of inner iterations (in parentheses).

	$p=4$	$p=9$	$p=16$	$p=25$
GMR6	78(68.4)	83(38.7)	145(31.4)	168(26.4)
GMR2	86(15.7)	118(15.7)	168(13.7)	192(10.9)
GMR1	139(13.6)	225(9.3)	287(7.1)	303(5.9)
RILUD	341(1)	291(1)	439(1)	437(1)

Note the large increase in the number of outer iterations incurred for GMR1 over GMR2, which helps to explain the faster time for GMR2. Apparently, an inner loop tolerance of 10^{-1} is insufficient for fast global convergence, yet is still a very expensive subdomain approximation. The RILUD approximation, on the other hand, though it gives the worst convergence rate of the outer loop, is very cheap to apply; in fact, cheap enough to make it the fastest method.

Figure 3.8 illustrates the speedup against the multiblock serial solution, obtained on the workstation cluster and on the Cray using GMR6, GMR2, GMR1 and RILUD subdomain approximations. We would expect nearly perfect speedup, especially for large problems, since the work required for preconditioning is proportional to the total number of unknowns, while the amount of communication is proportional to the length of subdomain interfaces. The observed speedup is quite good on the Cray; however, on the workstation cluster, especially for $p = 9$ the subdomain grid size needs to be quite large to obtain a high speedup. In any case we can conclude that if domain decomposition is going to be used anyway for geometric reasons or due to memory limitations, a speedup can be achieved by parallelization and subdomain approximation with an RILUD preconditioner.

Multiblock case, scaled problem size

Figure 3.9 shows a comparison of the parallel scalability of the domain decomposition method with approximate subdomain solution. The figure shows computation times on 1, 4 and 9 processors (1, 4, 9, 16 and 25 processors for the Cray T3E) with a fixed subdomain size of 120 × 120. A fixed number of outer iterations (30) were computed. Note that the method scales almost perfectly on the Cray for this range of processors. On the workstation cluster, the scaling is somewhat poorer, but reasonable.

3.5 Conclusions

For applications which require domain decomposition for some reason other than parallelism, it is possible to achieve a great reduction of computation time by solving subdomain problems approximately. A reasonable speedup with respect to the single block serial solution method is also attainable, particularly when using many processors of a massively parallel distributed memory machine. This speedup is less impressive when computing on a cluster of workstations, due to the increased communication latency.

In our experience, the best subdomain approximation method in parallel is a simple incomplete factorization restricted to the diagonal: the RILUD factorization. With this preconditioner used as a subdomain approximation, the approximate solves become so cheap (and yet sufficiently accurate) that they offset the increased number of global iterations resulting from inaccurate subdomain solution.

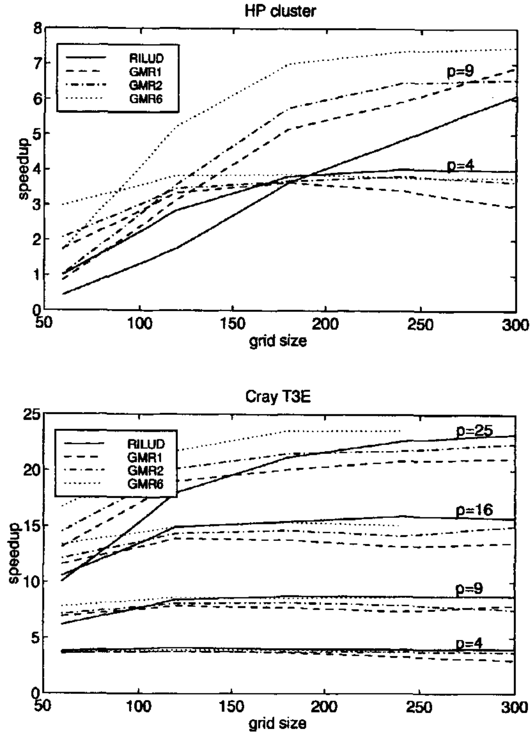

FIGURE 3.8: Speedup vs. multiblock solution on the cluster of workstations and the Cray T3E.

A performance model for the modified Gram-Schmidt, re-orthogonalized classical Gram-Schmidt, and Householder orthogonalization methods indicates that classical Gram-Schmidt and Householder require approximately the same amount of work and communication, making the classical Gram-Schmidt more attractive, since it is easier to implement and more stable. The Householder and re-orthogonalized Gram-Schmidt methods are most effective for relatively small problems: using nine processors, up to about 900 unknowns per processor for a Cray T3E, or 8000 unknowns per processor for a cluster of workstations. One promising area of application for these procedures is in long-time simulations of systems of this size.

Acknowledgements

The authors are indebted to Pieter Wesseling and an anonymous referee for critical remarks on the manuscript.

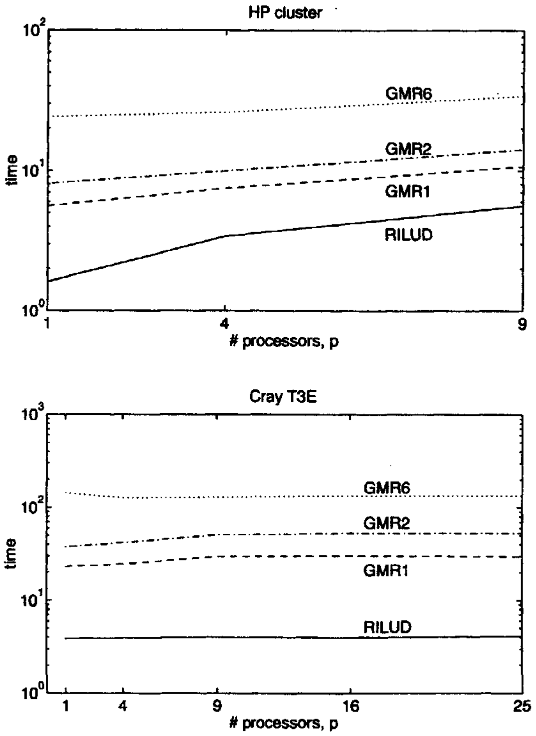

FIGURE 3.9: Computation time for fixed subdomain size of 120 × 120.

References

[1] O. Axelsson and G. Linskog. On the eigenvalue distribution of a class of preconditioning methods. *Numerische Mathematik*, 48:479–498, 1986.

[2] Z. Bai, D. Hu, and L. Reichel. A Newton-basis GMRES implementation. *IMA Journal of Numerical Analysis*, 14:563–581, 1994.

[3] Ake Björck. Solving linear least squares problems by Gram-Schmidt orthogonalization. *BIT*, 7:1–21, 1967.

[4] Christoph Börgers. The Neumann-Dirichlet domain decomposition method with inexact solvers on the subdomain. *Numerische Mathematik*, 55:123–136, 1989.

[5] E. Brakkee, A. Segal, and C. G. M. Kassels. A parallel domain decomposition algorithm for the incompressible Navier-Stokes equations. *Simulation Practice and Theory*, 3:185–205, 1995.

[6] E. Brakkee, C. Vuik, and P. Wesseling. Domain decomposition for the incompressible Navier-Stokes equations: Solving subdomain problems accurately and inaccurately. *International Journal for Numerical Methods in Fluids*, 26:1217–1237, 1998.

[7] Erik Brakkee. *Domain Decomposition for the Incompressible Navier-Stokes Equations.* PhD thesis, Delft University of Technology, P.O. Box 5031, 2600 GA Delft, The Netherlands, April 1996.

[8] Tony F. Chan and Tarek P. Mathew. Domain decomposition algorithms. In Arieh Iserles, editor, *Acta Numerica*, pages 61–143. Cambridge University Press, 1994.

[9] H. Cheng. On the effect of using inexact solvers for certain domain decomposition algorithms. *East-West J. Numer. Math.*, 2(4):257–284, 1994.

[10] E. de Sturler and H. A. van der Vorst. Reducing the effect of global communication in GMRES(m) and CG on parallel distributed memory computers. *Applied Numerical Mathematics*, 18:441–459, 1995.

[11] Stanley C. Eisenstat, Howard C. Elman, and Martin H. Schultz. Variational iterative methods for nonsymmetric systems of linear equations. *SIAM Journal on Numerical Analysis*, 20(2):345–357, April 1983.

[12] J. Erhel. A parallel GMRES version for general sparse matrices. *Electronic Transactions on Numerical Analysis (http://etna.mcs.kent.edu)*, 3:160–176, 1995.

[13] S. Goossens, E. Issman, G. Degrez, and D. Roose. Block $ILP^{-1}U(0)$ preconditioning for a GMRES based Euler/Navier-Stokes solver. In H. Liddell, A. Colbrook, B. Herzberger, and P. Sloot, editors, *High Performance Computing and Networking '96*, Lecture Notes in Computer Science 1067, pages 619–626. Springer-Verlag, 1996.

[14] Ivar Gustafsson. A class of first order factorization methods. *BIT*, 18:142–156, 1978.

[15] G. Haase, U. Langer, and A. Meyer. The approximate Dirichlet domain decomposition method. Part I: An algebraic approach. *Computing*, 47:137–151, 1991.

[16] G. Haase, U. Langer, and A. Meyer. The approximate Dirichlet domain decomposition method. Part I: Applications to 2nd-order elliptic B.V.P.s. *Computing*, 47:153–167, 1991.

[17] G. Haase, U. Langer, and A. Meyer. Domain decomposition preconditioners with inexact subdomain solvers. *Journal of Numerical Linear Algebra with Applications*, 1(1):27–41, 1991.

[18] R. W. Hockney and C. R. Jesshope. *Parallel Computers 2: Architecture, Programming and Algorithms.* Adam Hilger, Bristol, 1988.

[19] Walter Hoffman. Iterative algorithms for Gram-Schmidt orthogonalization. *Computing*, 41:335–348, 1989.

[20] W. Jalby and B. Philippe. Stability analysis and improvement of the block Gram-Schmidt algorithm. *SIAM Journal of Scientific and Statistical Computing*, 12(5):1058–1073, 1991.

[21] J. van Kan. A second-order accurate pressure-correction scheme for viscous incompressible flow. *SIAM Journal on Scientific and Statistical Computing*, 7(3):870–891, 1986.

[22] David E. Keyes. Aerodynamic applications of Newton-Krylov-Schwarz solvers. In S. M. Despande, S. S. Desai, and R. Narasimha, editors, *Fourteenth International Conference on Numerical Methods in Fluid Dynamics*, pages 1–20. Springer, Berlin, 1995.

[23] V. Kumar, A. Grama, A. Gupta, and G. Karypis. *Introduction to Parallel Computing; Design and Analysis of Algorithms.* Benjamin/Cummings, Redwood City, 1994.

[24] G. Li. A block variant of the GMRES method on massively parallel processors. *Parallel Computing 23*, 23:1005–1019, 1997.

[25] J. A. Meijerink and H. A. van der Vorst. An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix. *Mathematics of Computation*, 31:148–162, 1977.

[26] A. Meyer. A parallel preconditioned conjugate gradient method using domain decomposition and inexact solvers on each subdomain. *Computing*, 45:217–234, 1990.

[27] D. P. O'Leary and P. Whitman. Parallel QR factorization by Householder and modified Gram-Schmidt algortihms. *Parallel-Computing*, 16:99–112, 1990.

[28] Yousef Saad. A flexible inner-outer preconditioned GMRES algorithm. *SIAM Journal on Scientific and Statistical Computing*, 14:461–469, 1993.

[29] Yousef Saad and Martin H. Schultz. GMRES: A generalized minimum residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 7(3):856–869, July 1986.

[30] A. Segal, P. Wesseling, J. van Kan, C.W. Oosterlee, and K. Kassels. Invariant discretization of the incompressible Navier-Stokes equations in boundary-fitted co-ordinates. *International Journal for Numerical Methods in Fluids*, 15:411–426, 1992.

[31] R. B. Sidje. Alternatives for parallel Krylov subspace basis computation. *Numerical Linear Algebra with Applications*, 4(4):305–331, 1997.

[32] Barry F. Smith, Petter E. Bjørstad, and William D. Gropp. *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations.* Cambridge University Press, 1996.

[33] Kian H. Tan. *Local Coupling in Domain Decomposition.* PhD thesis, Utrecht University, P.O. Box 80010, 3508 TA Utrecht, The Netherlands, April 1996.

[34] Henk A. van der Vorst and C. Vuik. GMRESR: a family of nested GMRES methods. *Numerical Linear Algebra with Applications*, 1(4):369–386, 1994.

[35] C. Vuik. Further experiences with GMRESR. *Supercomputer*, 55:13–27, 1993.

[36] C. Vuik. New insights in GMRES-like methods with variable preconditioners. *Journal of Computational and Applied Mathematics*, 61:189–204, 1995.

[37] C. Vuik. Fast iterative solvers for the discretized incompressible Navier-Stokes equations. *International Journal for Numerical Methods in Fluids*, 22:195–210, 1996.

[38] Homer F. Walker. Implementation of the GMRES method using Householder transformations. *SIAM Journal on Scientific and Statistical Computing*, 9(1):152–163, 1988.

[39] P. Wesseling, A. Segal, and C.G.M. Kassels. Computing flows on general three-dimensional nonsmooth staggered grids. *J. Comp. Phys.*, 149:333–362, 1999.

[40] P. Wesseling, A. Segal, C.G.M. Kassels, and H. Bijl. Computing flows on general two-dimensional nonsmooth staggered grids. *Journal of Engineering Mathematics*, 34:21–44, 1998.

Chapter 4

Parallel block-preconditioned GCR for incompressible flow problems

Abstract. Efficient parallel algorithms are required to simulate incompressible turbulent flows in complex two- and three-dimensional domains. The incompressible Navier-Stokes equations are discretized in general coordinates on a structured grid. For a flow on a general domain we use an unstructured decomposition of the domain into subdomains of simple shape, with a structured grid inside each subdomain. We have developed a parallel block-preconditioned GCR method to solve the resulting systems of linear equations. The method can be smoothly varied between a coarse grain parallel method in which the subdomain problems are solved accurately with an inner iteration process and a fine grain parallel method when only a preconditioner is used to approximate the solution on the blocks. Parallel performance results for Boussinesq flow in a cavity are included.

4.1 Introduction

Efficient parallel algorithms are required to simulate incompressible turbulent flows in complex two- and three-dimensional domains. We consider the incompressible Navier-Stokes equations:

$$\frac{\partial \mathbf{u}}{\partial t} - \frac{1}{R_e}\Delta \mathbf{u} + \mathbf{u}\nabla \cdot \mathbf{u} + \nabla p = \mathbf{f},$$

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

where R_e is the Reynolds number. These equations are discretized in general coordinates using a staggered finite volume method on a structured grid, see [17, 28, 3, 27]. For a flow on a general domain we use an unstructured decomposition of the domain into subdomains of simple shape, with a structured grid inside each subdomain. We have developed a parallel block-preconditioned GCR method to solve the resulting systems of linear equations.

Let V^n and P^n represent the algebraic vectors containing velocity and pressure unknowns at time t^n, respectively. Application of the backward Euler method in time yields

$$\frac{V^{n+1} - V^n}{\Delta t} = F(V^n)V^{n+1} - GP^{n+1}, \tag{4.1}$$

$$DV^{n+1} = 0, \tag{4.2}$$

where (4.1) represents the discretized momentum equation and (4.2) represents the discretized incompressibility condition. The matrix F is the linearized spatial discretization of the convection and stress in the Navier-Stokes equations, G is the discretized gradient operator, and D is the discretized divergence operator. To solve (4.1) and (4.2) with the time-accurate pressure correction method [21], these equations are approximated by:

(Prediction)

$$\frac{V^* - V^n}{\Delta t} = F(V^n)V^* - GP^n \tag{4.3}$$

and

$$\frac{V^{n+1} - V^n}{\Delta t} = F(V^n)V^* - GP^{n+1}, \tag{4.4}$$

$$DV^{n+1} = 0. \tag{4.5}$$

Subtraction of (4.3) from (4.4) gives

$$\frac{V^{n+1} - V^*}{\Delta t} = -G(P^{n+1} - P^n). \tag{4.6}$$

Taking the discretized divergence of both sides of (4.6) and using (4.5) results in the pressure correction equation:

(Projection)

$$DG\Delta P = \frac{DV^*}{\Delta t}, \tag{4.7}$$

where $\Delta P = P^{n+1} - P^n$. After the pressure correction ΔP has been computed from (4.7), it is substituted into (4.6), which leads to:

(Correction)

$$V^{n+1} = V^* - \Delta t G \Delta P. \tag{4.8}$$

In summary, the pressure correction method consists of three steps: (i) computation of V^* from (4.3), (ii) computation of ΔP from (4.7) and computation of V^{n+1} from (4.8). The linear systems are solved by a Krylov subspace method with an ILU [23, 24] or a multigrid [30] preconditioner.

The linear systems (4.3) and (4.7) are solved with GCR [10, 20] using a block-diagonal preconditioner based on a nonoverlapping domain decomposition[9, 6, 29]. This allows us to handle more general domains with a structured discretization, by decomposing the domain into regions which are topologically similar to a square or cube, within each of which the grid is structured. The block-diagonal structure additionally facilitates parallelization, allowing us to handle very large domains in which memory limitations come into play.

Additionally, the independent blocks of the preconditioner can be solved as precisely as desired using an inner iteration procedure [5, 6]. Thus, our method can be smoothly varied between a coarse grain parallel method when the subdomain problems are solved accurately [4], to a fine grain parallel method when only one subdomain iteration is done in every domain decomposition iteration (compare [25]).

Efficient parallel implementation of GCR method requires, in addition to the preconditioner, a proper handling of the matrix vector multiplication and inner products. For a matrix vector product only nearest neighbor communications are required, which is efficient on most parallel computers. Inner products, on the other hand, require global communications; therefore, the focus has been on reducing the number of inner products [11, 18], overlapping inner product communications with computation [8], or increasing the number of inner products that can be computed with a single communication [2, 16].

The details of our domain decomposition algorithm are given in Section 4.2.1. The GCR method is summarized in Section 4.2.2, whereas various orthogonalization methods are discussed in Section 4.2.3. Speedup results are presented in Section 4.3.

4.2 The block-preconditioned GCR method

We begin by describing in detail our block-diagonal preconditioner, the Krylov subspace method used to accelerate the iterations, and its parallelization aspects, especially the question of an orthogonalization procedure.

4.2.1 The block Gauss-Jacobi preconditioner

The pressure correction algorithm, (4.3)-(4.8), is used for the solution of the Navier-Stokes equations on the global domain Ω. Let the domain be the union of M nonoverlapping subdomains Ω_m, $m = 1, \ldots, M$. The equations (4.3) and (4.7) are solved using domain decomposition. We require that the subdomains intersect regularly, i.e. the grid lines are continuous across block-interfaces. The correction of V^* (4.8) is independently carried out in all blocks.

When discretized on the global grid, both the momentum equation (4.3) and the pressure equation (4.7) can be written as a linear system

$$Av = f, \tag{4.9}$$

with either $A = S(V^n, P^n) := \frac{1}{\Delta t}I - F(V^n, P^n)$ and $v = V^*$ for the momentum equation or $A = DG$ and $v = \Delta P$ for the pressure correction equation. If we decompose A into blocks such that each block corresponds to all unknowns in a single subdomain, with a small modification for the momentum equation (see further on), then we get the block system

$$\begin{bmatrix} A_{11} & \dots & A_{1M} \\ \vdots & \ddots & \vdots \\ A_{M1} & \dots & A_{MM} \end{bmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_M \end{pmatrix} = \begin{pmatrix} b_1 \\ \vdots \\ b_M \end{pmatrix}. \tag{4.10}$$

In this system, one observes that the diagonal blocks A_{mm} express coupling among the unknowns defined on a common subdomain (Ω_m), whereas the off-diagonal blocks A_{mn}, $m \neq n$ represent coupling across subdomain boundaries. The only nonzero off-diagonal blocks are those corresponding to neighboring subdomains.

The unaccelerated domain decomposition iteration for Equation (4.9) is analogous to a classical iteration of the form:

$$v^{m+1} = v^m + K^{-1}(f - Av^m). \tag{4.11}$$

with the block Gauss-Jacobi method matrix K defined as

$$K = \begin{bmatrix} A_{11} & & \\ & \ddots & \\ & & A_{NN} \end{bmatrix}.$$

When (4.11) is used, systems of the form $Kv = r$ have to be solved. Since there is no overlap the diagonal blocks $A_{mm}v_m = r_m$, $m = 1, \dots, N$ can be solved in parallel. In our approach these systems are solved by an iterative method. An important point is the required tolerance of these inner iterations (see [5, 12]). Since the number of inner iterations may vary from one subdomain to another, and in each outer iteration, the effective operator $\hat{K}^{-1} \approx K^{-1}$ is nonlinear and varies in each outer iteration.

Our choice of approximate solution methods is motivated by the results obtained in [5] and [12]. where GMRES was used to approximately solve subdomain problems to within fixed tolerances of 10^{-4}, 10^{-3}, 10^{-2} and 10^{-1}. Additionally, a blockwise application of the RILU(D) preconditioner has been used [24].

We cannot apply the above described block Gauss-Jacobi algorithm directly to the momentum matrix S because the normal velocity components on the block interfaces belong to two blocks. Instead, we first augment the matrix S in the following way. It is sufficient to consider a decomposition into two blocks ($N = 2$). Let the velocity unknowns be divided into three sets:

1. The first set consists of velocities belonging to Block 1, excluding the normal velocities at the block interface.

2. The second set consists of the normal velocities at the interface.

3. The third set consists of velocities belonging to Block 2, excluding the normal velocities at the block interface

With respect to these sets of unknowns, the matrix $S(V^n, P^n)$ has the block form

$$S(V^n, P^n) = \begin{bmatrix} S_{11} & S_{12} & S_{13} \\ S_{21} & S_{22} & S_{23} \\ S_{31} & S_{32} & S_{33} \end{bmatrix}. \tag{4.12}$$

The system of equations $S(V^n, P^n)V^* = f$ can be augmented, doubling the interface unknowns to arrive at

$$\bar{S}(V^n, P^n)\bar{V}^* = \begin{bmatrix} S_{11} & S_{12} & 0 & S_{13} \\ S_{21} & S_{22} & 0 & S_{23} \\ S_{21} & 0 & S_{22} & S_{23} \\ S_{31} & 0 & S_{32} & S_{33} \end{bmatrix} \begin{bmatrix} \bar{V}_1^* \\ \bar{V}_2^* \\ \bar{V}_2'^* \\ \bar{V}_3^* \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ f_2 \\ f_3 \end{bmatrix}. \tag{4.13}$$

The solution of Equation (4.13) satisfies $\bar{V}_2^* = \bar{V}_2'^*$ whenever S_{22} is invertible (see [19]) and in this case, Equation (4.13) is equivalent to the original system of equations $S(V^n, P^n)V^* = f$. In view of Equation (4.10) we have

$$A_{11} = \begin{bmatrix} S_{11} & S_{12} \\ S_{21} & S_{22} \end{bmatrix} \text{ and } A_{22} = \begin{bmatrix} S_{22} & S_{23} \\ S_{32} & S_{33} \end{bmatrix}, \tag{4.14}$$

so that the domain decomposition for the momentum equation has been described.

4.2.2 The GCR method

The block Gauss-Jacobi iteration (4.11) described in Section 4.2.1 can be accelerated by a Krylov subspace method. To do so, we choose K^{-1} as preconditioner. Due to the approximate solution of the subdomain problems the effective preconditioner is nonlinear and varies in each outer iteration. We choose the GCR method [10, 20] because it can be used with a variable preconditioner, and furthermore is quite flexible for use with restart/truncation strategies when the number of iterations exceeds the prescribed limit n_{trunc}. In practice truncated GCR converges faster than restarted GCR. If the number of outer iterations is less than n_{trunc} an optimized version of the GCR method is used [22].

The preconditioned GCR method is given by:

```
Algorithm: GCR
Given: initial guess $x_0$
$r_0 = b - Ax_0$
for $k = 1, \ldots,$ convergence
    Solve $K\tilde{v} = r_{k-1}$ (approximately)
    $\tilde{q} = A\tilde{v}$
    $[q_k, v_k] =$ orthonorm $(\tilde{q}, \tilde{v}, q_i, v_i, i < k)$
    $\gamma = q_k^T r_{k-1}$
    Update: $x_k = x_{k-1} + \gamma v_k$
    Update: $r_k = r_{k-1} - \gamma q_k$
end
```

The vectors q_k and v_k are distributed over the processors in the same way as the solution vector x_k. All vectors $q_i, v_i, i \leq k$ are stored in memory. The function **orthonorm()** takes input vectors $\tilde{q}$ and $\tilde{v}$, orthogonalizes $\tilde{q}$ with respect to the q_i, $i < k$, and returns the modified vectors q_k such that $\|q_k\|_2 = 1$. In order to preserve the relation $\tilde{q} = A\tilde{v}$ equivalent operations are done with $\tilde{v}$.

Apart from the preconditioner, the main challenges to parallelization of GCR is parallel computation of the inner products, which require global communication and therefore do not scale. Much of the literature on parallel Krylov subspace methods and parallel orthogonalization methods has focused on performing several iterations before orthogonalizing, so that a number of vectors can be orthogonalized simultaneously with a single communication However, this is not possible using a preconditioner which varies in each iteration. For this reason, we need a method for orthogonalizing one new vector against an orthonormal basis of vectors.

4.2.3 Orthogonalization methods

A disadvantage of the modified Gram-Schmidt method in parallel is that the number of inner products increases proportionally to the iteration number and these inner products must be computed using successive communications. This is not the case if one uses the classical Gram-Schmidt method. In this algorithm all necessary inner products can be computed with a single global communication. Unfortunately, the classical Gram-Schmidt method is unstable with respect to rounding errors, so this method is rarely used. On the other hand, Hoffmann [14] gives experimental evidence indicating that a two-fold application of the classical Gram-Schmidt method *is* stable. Another method which has been suggested is the parallel implementation of Householder transformations, introduced by Walker [26]. Below we reformulate this method for GCR (see also [12]).

In the Householder orthogonalization we use the notion a_k to represent the kth column of a matrix A and $a^{(i)}$ to represent the ith component of a vector a. Let a matrix $A \in \mathbb{R}^{n \times m}$, $m \leq n$ with linearly independent columns be factored as QZ, where Q is orthogonal and Z is upper triangular. Then the kth column of A is given by $a_k = Qz_k$ and the columns of Q form an orthonormal basis for the span of the columns of A.

We construct Q as the product of a series of Householder reflections, $Q = P_1 \cdots P_m$, used to transform A into Z. The matrices $P_i = I - 2\frac{w_i w_i^T}{w_i^T w_i}$, with $w_i^{(j)} = 0$ for $j < i$ have the property: $P_i(P_{i-1} \cdots P_1)a_i = z_i$.

Suppose one has already produced k orthogonal basis vectors. To compute w_{k+1} one must first apply the previous reflections to a_{k+1} as described in [26]: $\tilde{a} = P_k \cdots P_1 a_{k+1} = (I - 2W_k L_k^{-1} W_k^T) a_{k+1}$, where W_k is the matrix whose columns are $w_1, \ldots, w_k$, and where

$$L_k = \begin{bmatrix} 1 & & & \\ 2w_2^T w_1 & 1 & & \\ \vdots & & \ddots & \\ 2w_k^T w_1 & \ldots & 2w_k^T w_{k-1} & 1 \end{bmatrix}.$$

Note especially that in the $(k+1)$th iteration one must compute the last row of L_k, which is the vector $(2w_k^T W_{k-1}, 1)$, as well as the vector $W_k^T a_{k+1}$. This requires $2k-1$ inner products, but they may all be computed using only a single global communication.

Let $\hat{a}$ be the vector obtained by setting the first k elements of $\tilde{a}$ to zero. The vector w_{k+1} is chosen as: $w_{k+1} = \hat{a} + \mathrm{sign}\,(\hat{a}^{(k+1)})\|\hat{a}\|_2 e_{k+1}$. In practice, the vectors w_k are normalized to length one. The length of w_{k+1} can be expressed as $\|w_{k+1}\|_2 = \sqrt{2\alpha^2 - 2\alpha\hat{a}^{(k+1)}}$ where $\alpha = \mathrm{sign}\,(\hat{a}^{(k+1)})\|\hat{a}\|_2$. The $(k+1)$th column of Q is the new orthonormal basis vector:

$$q_{k+1} = \frac{1}{\alpha}\left[a_{k+1} - \sum_{i=1}^{k} \tilde{a}^{(i)} q_i\right].$$

Within the GCR algorithm, the linear combination with the same coefficients must be applied to the v_i to obtain v_{k+1}.

In Table 4.1 we summarize the round-off properties and the amount of work and communication for the following orthogonalization methods (for details see [12]):

- Classical Gram-Schmidt (CGS)
- Reorthogonalized Classical Gram-Schmidt (RCGS)
- Modified Gram-Schmidt (MGS)
- Householder (HH)

Comparing the costs we expect that the wall-clock time for RCGS and HH are comparable. When communication is slow (large latency) compared to computation, one expects that these methods are faster than MGS, with of course a preference for RCGS. Otherwise, when computational costs dominate, MGS is the fastest method because it requires fewer floating point operations.

	round-off	daxpy	ddot	communications
CGS	bad	2k	k	1
MGS	good	2k	k	k
RCGS	good	3k	2k	2
HH	good	3k	2k	3

TABLE 4.1: Properties of the various orthogonalization methods

4.3 Numerical experiments

In this section we illustrate the parallel performance of the block Gauss-Jacobi preconditioned GCR method when implemented within the Navier-Stokes software DeFT [27]. Numerical experiments were performed on a network of workstations (NOW) consisting of Hewlett-Packard 700-series machines connected by a 10 Mb Ethernet and on a Cray T3E.

The test problem considered was a two-dimensional Boussinesq flow [7] on $(0,1)\times(0,1)$. The governing equations are given by

$$\frac{\partial \mathbf{u}}{\partial t} - \frac{1}{R_e}\Delta \mathbf{u} + \mathbf{u}\nabla\cdot\mathbf{u} + \nabla p = \mathbf{g}\frac{G_r}{R_e^2}T, \tag{4.15}$$

$$\nabla\cdot\mathbf{u} = 0, \tag{4.16}$$

$$\frac{dT}{dt} - \frac{1}{R_e P_r}\Delta^2 T + \mathbf{u}\cdot\nabla T = 0, \tag{4.17}$$

with $\mathbf{g} = (0,-1)$ and boundary conditions $\mathbf{u}(0,y) = \mathbf{u}(1,y) = \mathbf{u}(x,0) = \mathbf{u}(x,1) = \mathbf{0}$, $T(0,y) = 1$, $T(1,y) = 0$, and $\partial T/\partial y(x,0) = \partial T/\partial y(x,1) = 0$. The Reynolds, Prandtl and Grashof numbers were taken to be $R_e = 1$, $P_r = 0.71$ (air) and $G_r = 1500$, respectively. The simulation was carried out for 10 timesteps of size $\Delta t = 0.05$ to diffuse the influence of start-up latencies. It is known that due to the temperature difference a circulating flow arises, with the number of vortices depending on the Grashof number. This recirculation makes the momentum equation (4.15) and heat transport equation (4.17) relatively difficult to solve. Since the domain is rectangular, it is easily decomposed into various block configurations.

In the inner iteration process, blocks were solved to various accuracies using GMRES with a restart of 40, preconditioned with the relaxed incomplete factorization, RILU(α), of [1] using a relaxation parameter $\alpha = 0.975$ for the pressure correction equation (4.7) and $\alpha = 1$ for the momentum and transport equations. In the extreme case, we perform no GMRES iterations and use only the RILU preconditioner on the blocks. For the outer iterations, GCR was used with a Krylov subspace of dimension 25 and employing the Jackson and Robinson truncation strategy [15, 22].

The timings listed in this section are wall-clock times obtained with MPI timing routines, and indicate the time spent in the linear solver part of the code. In particular, they do not include time required to construct the matrices.

In all of our tests, we observed very similar behavior for the transport equation (4.17) as for the pressure equation (4.7), so we will neglect the discussion of the transport equation in the ensuing.

4.3.1 Comparison with diagonal scaling

As a basis for comparison of the effectiveness of the block preconditioner, we ran a few tests using a simple diagonal scaling (Gauss-Jacobi) preconditioner. This preconditioner is very popular in a parallel computing environment. Table 4.2 gives wall-clock times and iteration counts using both preconditioners. The table indicates that the number of iterations required for convergence with diagonal preconditioning is quite large and increases drastically as the grid is refined. Furthermore the block Gauss-Jacobi preconditioner needs much less wall-clock time than the Gauss-Jacobi preconditioner.

TABLE 4.2: Wall-clock time and iteration counts given in parentheses for a Gauss-Jacobi and block Gauss-Jacobi preconditioning

		Gauss-Jacobi		block Gauss-Jacobi	
blocks	subgrid	Momentum	Pressure	Momentum	Pressure
2 × 2	24 × 24	13.8 (119)	9.0 (144)	4.8 (39)	2.6 (38)
	60 × 60	159 (301)	101 (390)	62.6 (91)	21.2 (69)
3 × 3	24 × 24	25.2 (180)	19.7 (226)	8.7 (60)	6.1 (64)

4.3.2 Comparison with serial block-preconditioner

To measure the cost of parallelization, we compare the parallel and sequential computation times using the block Gauss-Jacobi preconditioners. Tables 4.3 and 4.4 give the speedup factors on a Cray T3E for the momentum and pressure equations, respectively, using the approximate solvers or the RILU preconditioner on the blocks. The subdomain approximations will be denoted as follows:

- GMR6 = restarted GMRES with a tolerance of 10^{-6},
- GMR2 = restarted GMRES with a tolerance of 10^{-2},
- GMR1 = restarted GMRES with a tolerance of 10^{-1},

- RILU = one application of an RILU preconditioner.

The trends are as expected: when the blocks are solved very accurately, the relative cost of communication to computation is low, giving a high speedup in parallel; whereas for the less accurate approximations, the communications are relatively more expensive, and a lower speedup is observed. In general, the parallel efficiency is quite high for a small number of blocks but decreases as the number of blocks is increased. The speedups are higher for the momentum equation than for the pressure equation.

TABLE 4.3: Attained speedups over sequential implementation (Momentum equation, 24 × 24 subgrid resolution)

blocks	GMR6	GMR2	GMR1	RILU(1)
4	3.7	3.6	3.6	3.4
9	8.1	7.5	7.4	7.0
16	14.0	12.9	12.6	12.2
25	18.4	17.1	16.6	16.4

TABLE 4.4: Attained speedups over sequential implementation (Pressure equation, 24 × 24 subgrid resolution)

blocks	GMR6	GMR2	GMR1	RILU(0.95)
4	3.2	3.0	2.9	2.6
9	6.4	5.9	5.4	5.0
16	10.5	9.3	9.2	9.3
25	14.2	9.9	9.6	12.1

4.3.3 Scalability comparison

Fixed problem size

In this section we compare the parallel computation times for a fixed problem size on a 120 × 120 grid. The grid is decomposed into 2 × 2, 3 × 3, 4 × 4 and 5 × 5 subdomains. Tables 4.5 and 4.6 give timing results for the momentum and pressure equations. The number of outer iterations required in the final timestep is given in parentheses.

The single block solution times are listed in each table for reference. The number of necessary outer iterations increases severely in the multiblock case as compared to the single block case of only one iteration. This initial loss of convergence rate can only be offset in the case of the momentum equation by using very rough approximations on the blocks and many processors. For the pressure equation, some speedup can already be obtained with only 4 blocks.

TABLE 4.5: Scalability study on 120 × 120 grid (Momentum equation)

	Single block solution time = 21.4 (1)			
blocks	GMR6	GMR2	GMR1	RILU(1)
2 × 2	200. (36)	72.7 (38)	56.1 (58)	62.1 (78)
3 × 3	74.4 (50)	34.9 (52)	28.7 (62)	30.8 (77)
4 × 4	42.4 (56)	22.9 (39)	20.2 (69)	19.3 (80)
5 × 5	32.4 (51)	18.3 (51)	16.7 (54)	17.0 (97)

TABLE 4.6: Scalability study on 120 × 120 grid (Pressure equation)

	Single block solution time = 30.8 (1)			
blocks	GMR6	GMR2	GMR1	RILU(0.95)
2 × 2	71.5 (29)	40.4 (29)	44.1 (30)	20.5 (66)
3 × 3	35.9 (33)	22.8 (33)	23.7 (34)	15.8 (87)
4 × 4	22.9 (39)	21.3 (65)	17.5 (40)	13.0 (103)
5 × 5	19.5 (62)	19.0 (76)	21.3 (91)	14.5 (116)

Fixed subdomain size

It is often argued that a better measure of the effectiveness of a parallel algorithm is obtained by fixing the per-processor problem size while increasing the number of processors [13]. In this section we therefore fix the subdomain grid at 24 × 24, and the domain decomposition is increased from a single block to a 5×5 block decomposition. In tables 4.7 and 4.8 we list the wall-clock times for the momentum and pressure equations, respectively. For perfect scaling, the wall-clock time would be constant, independent of the number of blocks. Given in parentheses are the number of outer iterations required in the final time step. For a fixed block size we observe for the momentum equation that the computation time scales roughly as the square root of the number of blocks. For the pressure equation the scaling is somewhat poorer, especially for the 5 × 5 block decomposition. For both equations there is a large increase in the number of outer iterations.

TABLE 4.7: Scalability study with fixed block size (Momentum equation)

blocks	GMR6	GMR2	GMR1	RILU(1)
2 × 2	10.5 (22)	5.7 (22)	4.9 (24)	4.7 (36)
3 × 3	16.4 (39)	9.3 (39)	8.3 (43)	8.5 (50)
4 × 4	22.3 (50)	12.7 (51)	11.6 (57)	11.6 (66)
5 × 5	32.4 (51)	18.3 (51)	16.7 (54)	17.0 (97)

TABLE 4.8: Scalability study with fixed block size (Pressure equation)

blocks	GMR6	GMR2	GMR1	RILU(0.95)
2×2	4.2 (20)	2.7 (23)	2.8 (20)	2.6 (37)
3×3	8.6 (27)	6.3 (27)	6.5 (28)	6.0 (62)
4×4	12.8 (36)	10.3 (36)	10.2 (37)	8.9 (90)
5×5	19.5 (62)	19.0 (76)	21.3 (91)	14.5 (116)

4.3.4 Orthogonalization methods

In this section we compare parallel performances of the modified Gram-Schmidt (MGS), Householder (HH), and reorthogonalized classical Gram-Schmidt (RCGS) processes on a NOW and on a Cray T3E. First we compare these methods for an artificial test problem. Thereafter we make a comparison for the Boussinesq problem.

In our first experiment the wall-clock times in the orthogonalization part are measured when 60 GCR iterations are performed. In Figure 4.1 the parameters

$$\mathcal{F}_{\mathrm{HH}} = \frac{\text{orthog. time MGS}}{\text{orthog. time HH}} \text{ and } \mathcal{F}_{\mathrm{RCGS}} = \frac{\text{orthog. time MGS}}{\text{orthog. time RCGS}}$$

are plotted as functions of n. In each subdomain an $n \times n$ grid is used. The number of subdomains is equal to the number of processors. On the workstation cluster (HH) and (RCGS) are only advantageous when the number of unknowns is less than 3600 on 4 processors and less than 6400 on 9 processors. On the Cray T3E, the number of unknowns per processor should be fewer than 1000 for 9 or even 25 processors. For larger problems the smaller amount of work involved in modified Gram-Schmidt orthogonalization outweighs the increased communication cost. Furthermore we observe that RCGS is somewhat more efficient than HH. Therefore we have not implemented the Householder orthogonalization in our Navier-Stokes solver.

Finally we report the total wall-clock time spent solving the linear systems originating from the two-dimensional Boussinesq flow problem. We consider the case for which orthogonalization is most likely to be a factor, i.e. relatively small blocks approximated by the RILU preconditioner. Since the approximate block solver is cheaper in this case, the communication costs weigh more heavily.

Table 4.9 compares times obtained on a Cray T3E. We see that the orthogonalization time is actually negligible on the Cray, so that neither of the strategies (MGS/RCGS) provides a significant advantage.

Table 4.10 presents analogous results on the NOW. For the 4-block decomposition, the workstations were directly connected by Ethernet, whereas for the 9-block decomposition, the workstations were located at different points on the local network, such that some messages had to pass through routers. Due to the relatively low communication bandwidth of the Ethernet, the inner product communications become an expensive

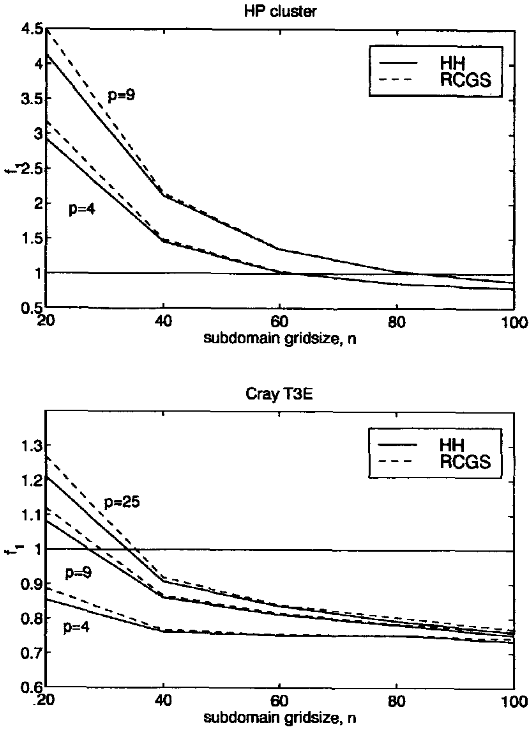

FIGURE 4.1: Measured speedup with Householder (HH) orthogonalization and reorthogonalized classical Gram-Schmidt (RCGS) with respect to modified Gram-Schmidt (MGS)

part of the computation, and a good speedup can be achieved by using reorthogonalized classical Gram-Schmidt.

4.4 Conclusions

In this paper we have presented parallel performance results for a block Gauss-Jacobi preconditioned GCR method for the Navier-Stokes equations. We summarize these results in the following remarks:

- The block Gauss-Jacobi preconditioner is perfectly parallel and gives better performance than a simple diagonal scaling which is also perfectly parallel.

- The convergence rate degrades substantially as the number of blocks is increased from one, but less appreciably thereafter. Current research into using overlap or multilevel techniques promises to improve this behavior.

TABLE 4.9: Comparison of computation times on a Cray T3E using MGS and RCGS orthogonalization processes (24 × 24 subgrid resolution, RILU subdomain approximation)

	MGS		RCGS	
blocks	Momentum	Pressure	Momentum	Pressure
2 × 2	4.7	2.6	4.7	2.6
3 × 3	8.5	6.0	8.5	5.5
4 × 4	11.6	8.9	11.7	8.4

TABLE 4.10: Comparison of computation times on a NOW using MGS and RCGS orthogonalization processes (24 × 24 subgrid resolution, RILU subdomain approximation)

	MGS		RCGS	
blocks	Momentum	Pressure	Momentum	Pressure
2 × 2	38.6	36.5	23.3	17.0
3 × 3	563	799	164	236

- It is sometimes advantageous to use the reorthogonalized classical Gram-Schmidt process on workstation clusters, particularly when the number of workstations is large and the network is slow.
- Parallelization of a multi-block problem leads to good speedups; however using this kind of domain decomposition simply for exploiting a parallel machine leads to only a modest decrease of wall-clock time.

Acknowledgement

The authors thank HPaC for providing computing facilities on the Cray T3E.

References

[1] O. Axelsson and G. Lindskog. On the rate of convergence of the preconditioned gradient method. *Num. Math.*, 48:499–523, 1986.

[2] Z. Bai, D. Hu, and L. Reichel. A Newton-basis GMRES implementation. *IMA J. Num. Anal.*, 14:563–581, 1994.

[3] H. Bijl and P. Wesseling. A unified method for computing incompressible and compressible flows in boundary-fitted coordinates. *J. Comp. Phys.*, 141:153–173, 1998.

[4] E. Brakkee, A. Segal, and C.G.M. Kassels. A parallel domain decomposition algorithm for the incompressible Navier-Stokes equations. *Simulation Practice and Theory*, 3:185–205, 1995.

[5] E. Brakkee, C. Vuik, and P. Wesseling. Domain decomposition for the incompressible Navier-Stokes equations: solving subdomain problems accurately and inaccurately. *Int. J. for Num. Meth. Fluids*, 26:1217–1237, 1998.

[6] J.H. Bramble, J.E. Pasciak, and A.T. Vassilev. Analysis of non-overlapping domain decomposition algorithms with inexact solves. *Math. Comp.*, 67:1–19, 1998.

[7] G. de Vahl Davis and I.P. Jones. Natural convection in a square cavity: a comparison exercise. *Int. J. Num. Meth. Fluids*, 3:227–248, 1983.

[8] E. de Sturler and H. A. van der Vorst. Reducing the effect of global communication in GMRES(m) and CG on parallel distributed memory computers. *Appl. Num. Math.*, 18:441–459, 1995.

[9] M. Dryja and O.B. Widlund. Domain decomposition algorithms with small overlap. *SIAM J. Sci. Comp.*, 15:604–620, 1994.

[10] S.C. Eisenstat, H.C. Elman, and M.H. Schultz. Variational iterative methods for nonsymmetric systems of linear equations. *SIAM J. Num. Anal.*, 20:345–357, 1983.

[11] J. Erhel. A parallel GMRES version for general sparse matrices. *Electronic Transactions on Numerical Analysis (*`http://etna.mcs.kent.edu`*)*, 3:160–176, 1995.

[12] J. Frank and C. Vuik. Parallel implementation of a multiblock method with approximate subdomain solution. *Appl. Num. Math.*, 30:403–423, 1999.

[13] J.L. Gustafson. Reevaluating Amdahl's law. *Comm. ACM*, 31:532–533, 1988.

[14] Walter Hoffmann. Iterative algorithms for Gram-Schmidt orthogonalization. *Computing*, 41:335–348, 1989.

[15] J.P. Jackson and P.C. Robinson. A numerical study of various algorithms related to the preconditioned conjugate gradient method. *Int. Num. Meth. Engng*, 21:1315–1338, 1985.

[16] G. Li. A block variant of the GMRES method on massively parallel processors. *Parallel Computing*, 23:1005–1019, 1997.

[17] A. Segal, P. Wesseling, J. Kan, C.W. Oosterlee, and K. Kassels. Invariant discretization of the incompressible nawier-stokes equations in boundary fitted coordinates. *Int. J. for Num. Meth. Fluids*, 15:411–426, 1992.

[18] R. B. Sidje. Alternatives for parallel Krylov subspace basis computation. *Num. Lin. Alg. Appl.*, 4(4):305–331, 1997.

[19] W. Tang. Generalized Schwarz splittings. *SIAM J. Sci. Stat. Comput*, 13:573–595, 1992.

[20] H.A. van der Vorst and C. Vuik. GMRESR: a family of nested GMRES methods. *Num. Lin. Alg. Appl.*, 1:369–386, 1994.

[21] J. van Kan. A second-order accurate pressure-correction scheme for viscous incompressible flow. *SIAM J. Sci. Stat. Comput.*, 7:870–891, 1986.

[22] C. Vuik. Further experiences with GMRESR. *Supercomputer*, 55:13–27, 1993.

[23] C. Vuik. Solution of the discretized incompressible Navier-Stokes equations with the GMRES method. *Int. J. for Num. Meth. Fluids*, 16:507–523, 1993.

[24] C. Vuik. Fast iterative solvers for the discretized incompressible Navier-Stokes equations. *Int. J. for Num. Meth. Fluids*, 22:195–210, 1996.

[25] C. Vuik, R.R.P. van Nooyen, and P. Wesseling. Parallelism in ILU-preconditioned GMRES. *Parallel Computing*, 24:1927–1946, 1998.

[26] Homer F. Walker. Implementation of the GMRES method using Householder transformations. *SIAM Journal on Scientific and Statistical Computing*, 9(1):152–163, 1988.

[27] P. Wesseling, A. Segal, and C.G.M. Kassels. Computing flows on general three-dimensional nonsmooth staggered grids. *J. Comp. Phys.*, 149:333–362, 1999.

[28] P. Wesseling, A. Segal, C.G.M. Kassels, and H. Bijl. Computing flows on general two-dimensional nonsmooth staggered grids. *J. Eng. Math.*, 34:21–44, 1998.

[29] J. Xu and J. Zou. Some nonoverlapping domain decomposition methods. *SIAM Review*, 40:857–914, 1998.

[30] S. Zeng, C. Vuik, and P. Wesseling. Numerical solution of the incompressible Navier-Stokes equations by Krylov subspace and multigrid methods. *Adv. Comp. Math.*, 4:27–49, 1995.

Chapter 5

Parallel Iteration of the Extended Backward Differentiation Formulas

Abstract. The extended backward differentiation formulas (EBDFs) and their modified form (MEBDF) were proposed by Cash in the 1980s for solving initial-value problems (IVPs) for stiff systems of ordinary differential equations (ODEs). In a recent performance evaluation of various IVP solvers, including a variable-stepsize-variable-order implementation of the MEBDF method of Cash, it turned out that MEBDF often outperforms codes such as RADAU5, DASSL and VODE. This motivated us to look at possible parallel implementations of the MEBDF method. Each MEBDF step essentially consists of successively solving three nonlinear systems by means of modified Newton iteration using the same Jacobian matrix. In a direct implementation of the MEBDF method on a parallel computer system, the only scope for (coarse grain) parallelism consists of a number of parallel vector updates. However, all forward-backward substitutions and all righthand side evaluations have to be done in sequence. In this paper, our starting point is the original (unmodified) EBDF method. As a consequence, two different Jacobian matrices are involved in the modified Newton method, but on a parallel computer system, the effective Jacobian-evaluation and the LU-decomposition costs are not increased. Furthermore, we consider the simultaneous solution, rather than the successive solution, of the three nonlinear systems, so that in each iteration the forward-backward substitutions and the righthand side evaluations can be done concurrently. A mutual comparison of the performance of the parallel EBDF approach and the MEBDF approach shows that we can expect a speedup factor of about 2 on 3 processors.

5.1 Introduction

The extended backward differentiation formulas (EBDFs) were proposed by Cash[2] in 1980 for solving initial-value problems (IVPs) for stiff systems of ordinary differential equations (ODEs)

$$\frac{dy}{dt} = f(y), \; y, f \in \mathcal{R}^d, \; t \geq t_0. \tag{5.1}$$

To conserve notation we will consider the autonomous problem, but the results of this paper can be trivially extended to derivatives with explicit dependence on time i.e. $f = f(t, y)$. Each EBDF step essentially consists of successively solving three nonlinear systems by means of (modified) Newton iteration. Since two different Jacobian matrices are involved, the method needs two different LU decompositions after each Jacobian update or change of stepsize. In order to reduce the LU costs, Cash[3] modified the EBDF methods (MEBDF methods) such that only one LU decomposition is required.

In a recent performance evaluation [11] of various IVP solvers, including a variable-step-variable-order implementation of the MEBDF method due to Cash, it turned out that the MEBDF code often performs more efficiently than codes like RADAU5 [9], DASSL[12] and VODE[1]. This motivated us to look at possible parallel implementations of the MEBDF method.

In a direct implementation of MEBDF on a parallel computer system, the only scope for (coarse grain) parallelism consists of a number of parallel vector updates. However, all forward- backward substitutions and all righthand side evaluations have to be done in sequence. In this paper, our starting point is the original (unmodified) EBDF method. As a consequence, two different Jacobian matrices are involved in the modified Newton method, but on a parallel computer system, the effective costs of the Jacobian-evaluations and the LU-decompositions are not increased. Furthermore, we consider the simultaneous solution, rather than the successive solution, of the three nonlinear systems, so that in each iteration the forward-backward substitutions and the righthand side evaluations can be done concurrently. A mutual comparison of the performance of the parallel EBDF and MEBDF approaches shows that we can expect a speedup factor of about 2 on 3 processors.

5.2 The EBDF and MEBDF methods of Cash

The EBDF method of Cash[2] is based on the formula

$$y_{n+1} = a_1 y_n + a_2 y_{n-1} + \cdots + a_k y_{n-k+1} + hb_0 f(y_{n+1}) + hb_1 f(y_{n+2}) \tag{5.2}$$

for computing an approximation y_{n+1} to the exact solution $y(t_{n+1})$ of (5.1). Here, y_{n+2} is an approximation to $y(t_{n+2})$ obtained by some predictor formula and the coefficients a_i and b_i are determined by imposing the conditions for order $k+1$ accuracy. Cash used the standard (implicit) BDF as a predictor:

$$\begin{aligned} u_{n+1} &= \bar{a}_1 y_n + \bar{a}_2 y_{n-1} + \cdots + \bar{a}_k y_{n-k+1} + h\bar{b}_0 f(u_{n+1}), \\ u_{n+2} &= \bar{a}_1 u_{n+1} + \bar{a}_2 y_n + \cdots + \bar{a}_k y_{n-k+2} + h\bar{b}_0 f(u_{n+2}), \end{aligned} \tag{5.3a}$$

to obtain an approximation u_{n+2} for the 'future' value y_{n+2}. Thus, y_{n+1} is computed from the equation

$$y_{n+1} = a_1 y_n + a_2 y_{n-1} + \cdots + a_k y_{n-k+1} + hb_0 f(y_{n+1}) + hb_1 f(u_{n+2}). \tag{5.3b}$$

The coefficients $\bar{a}_i$ and $\bar{b}_0$ are the BDF coefficients. The internal vectors u_{n+1} and u_{n+2} defined by (5.3a) have order of accuracy k and the external (or output) vector y_{n+1}

defined by (5.3b) has order of accuracy $k+1$. Hence the stage order s equals k and the actual order p equals $k+1$. Furthermore, $\{(5.3a),(5.3b)\}$ possesses a considerably larger stability region than the classical BDF method of order $p=k+1$. This can be explained by observing that the underlying corrector formula (5.2) is much more stable than the classical BDF (for $k>1$). For future reference, the coefficients $\{\bar{a}_i, \bar{b}_0\}$ and $\{a_i, b_0, b_1\}$ are given in the Tables 5.1 and 5.2 for $k=2,\ldots,5$. The MEBDF method arises from the EBDF method $\{(5.3a),(5.3b)\}$ by replacing (5.3b) with the formula (see [3, 8])

$$\begin{aligned} y_{n+1} = a_1 y_n + a_2 y_{n-1} + \cdots + a_k y_{n-k+1} + h\bar{b}_0 f(y_{n+1}) \\ +h(b_0 - \bar{b}_0) f(u_{n+1}) + h b_1 f(u_{n+2}). \end{aligned} \tag{5.3c}$$

The advantage is that the modified Newton iteration of the subsystems (5.3a) and (5.3b) can use the same LU-decomposition. Furthermore, the order of accuracy is not affected and the stability regions are even slightly larger than for the EBDF methods.

Note that in $\{(5.3b),(5.3c)\}$ each of the required derivatives $f(u_{n+1})$ and $f(u_{n+2})$ can be computed either with an additional function evaluation or by solving the corresponding formula (5.3a) for the (converged) derivative as a linear combination of back-values, whichever is cheaper [3].

TABLE 5.1: Coefficients $\{\bar{a}_i, \bar{b}_0\}$ in the BDF formulas (5.3a).

k	$\bar{a}_1\delta$	$\bar{a}_2\delta$	$\bar{a}_3\delta$	$\bar{a}_4\delta$	$\bar{a}_5\delta$	$\bar{b}_0\delta$	δ
2	4	-1				2	3
3	18	-9	2			6	11
4	48	-36	16	-3		12	25
5	300	-300	200	-75	12	60	137

TABLE 5.2: Coefficients $\{a_i, b_0, b_1\}$ in the EBDF and MEBDF formulas (5.3b) and (5.3c).

k	$a_1\delta$	$a_2\delta$	$a_3\delta$	$a_4\delta$	$a_5\delta$	$b_0\delta$	$b_1\delta$	δ
2	28	-5				22	-4	23
3	279	-99	17			150	-18	197
4	4008	-2124	728	-111		1644	-144	2501
5	26550	-18700	9600	-2925	394	8820	-600	14919

5.2.1 The implicit relations

The EBDF and MEBDF methods are implicit in u_{n+1}, u_{n+2} and y_{n+1}, and use the back values $y_{n-k+1},\ldots,y_n$ as input. Let us define the stage vector Y_{n+1} and the input vector V_n according to

$$Y_{n+1} = \begin{pmatrix} u_{n+1} \\ u_{n+2} \\ y_{n+1} \end{pmatrix}, \quad V_n = \begin{pmatrix} y_{n-k+1} \\ \vdots \\ y_n \end{pmatrix}.$$

Then, using tensor notation, both the EBDF method {(5.3a), (5.3b)} and the MEBDF method {(5.3a), (5.3c)} can be represented in the compact form

$$(B \otimes I)Y_{n+1} - h(C \otimes I)F(Y_{n+1}) = (E \otimes I)V_n, \tag{5.4}$$

Here, $\otimes$ denotes the Kronecker product, h the stepsize $t_{n+1} - t_n$, and $F(Y_{n+1})$ contains the righthand sides $f(u_{n+1})$, $f(u_{n+2})$, $f(y_{n+1})$. We denote the identity matrix by I, and its dimension will always be clear from the context. In the EBDF case, B, C and E are defined by

$$B := \begin{bmatrix} 1 & 0 & 0 \\ -\bar{a}_1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, C := \begin{bmatrix} \bar{b}_0 & 0 & 0 \\ 0 & \bar{b}_0 & 0 \\ 0 & b_1 & b_0 \end{bmatrix}, E := \begin{bmatrix} \bar{a}_k & \bar{a}_{k-1} & \cdots & \bar{a}_1 \\ 0 & \bar{a}_k & \cdots & \bar{a}_2 \\ a_k & a_{k-1} & \cdots & a_1 \end{bmatrix}. \tag{5.5}$$

In the MEBDF case, the last row of the matrix C changes to $\left(b_0 - \bar{b}_0, b_1, \bar{b}_0\right)$.

5.2.2 Iteration processes

Instead of solving for the unknown components u_{n+1}, u_{n+2} and y_{n+1} of Y_{n+1} *sequentially*, as was the original approach of Cash, we here consider an approach where these components are solved *simultaneously*, that is, the subsystems in the EBDF or MEBDF method are solved simultaneously. As we shall see below, one option in this approach is approximating the matrix $B^{-1}C$ by a diagonal matrix. The relative error of the diagonal approximation will be smaller if the diagonal elements of $B^{-1}C$ are large compared to the off-diagonal elements. For this reason, we choose the EBDF method, rather than the MEBDF method, as our starting point, because the additional nonzero off-diagonal element of the MEBDF matrix is taken out of the diagonal such that the row sum remains the same.

Premultiplying (5.4) by $B^{-1} \otimes I$, we can rewrite it in the form

$$R_n(Y_{n+1}) = 0, \tag{5.6}$$

$$R_n(Y) := Y - h(A \otimes I)F(Y) - (B^{-1}E \otimes I)V_n, \tag{5.7}$$

$$A := B^{-1}C = \begin{bmatrix} \bar{b}_0 & 0 & 0 \\ \bar{a}_1\bar{b}_0 & \bar{b}_0 & 0 \\ 0 & b_1 & b_0 \end{bmatrix}.$$

Let us iterate the system of implicit relations $R_n(Y_{n+1}) = 0$ by the Newton type method

$$(I - A^* \otimes hJ_{n+1})(Y^{(j)} - Y^{(j-1)}) = -R_n(Y^{(j-1)}), \; j = 1, \ldots, m, \tag{5.8}$$

where A^* is a suitably chosen matrix, $Y^{(0)}$ is an initial approximation to Y_{n+1}, and J_{n+1} is an approximation to the Jacobian matrix of the righthand side function in (5.1) at t_{n+1} (note that in the preceding timestep an approximation to y_{n+1} has been computed, which can be used to update the Jacobian if necessary). Suppose that the first and third component of $Y^{(0)}$ are defined by the second component of the stage vector Y_n computed in the preceding step, and that the second component of $Y^{(0)}$ is obtained by extrapolation of the most recent approximations available at the points $t_{n+1}, t_n, \ldots, t_{n-k+1}$. Then, $Y^{(0)}$ has order of accuracy $p = k$ and is expected to be an excellent initial approximation to Y_{n+1}. Moreover, the computational costs are negligible.

It is tempting to set $A^* = A$ resulting in the familiar (modified) Newton method and to try to diagonalize (5.8) by a Butcher similarity transformation $\tilde{Y}^{(j)} = (Q^{-1} \otimes I)Y^{(j)}$ such that the matrix $Q^{-1}AQ$ is diagonal. Unfortunately, the matrix A is defective, so that this does not work. However, if we approximate A by the matrix

$$A^* = \begin{bmatrix} \bar{b}_0 & 0 & 0 \\ 0 & \bar{b}_0 & 0 \\ c_1 & c_2 & b_0 \end{bmatrix}, \tag{5.9a}$$

then diagonalization is possible. It can be shown that

$$Q := \begin{bmatrix} q_1 & 0 & 0 \\ q_0 & q_2 & 0 \\ \frac{c_1 q_1 + c_2 q_2}{\bar{b}_0 - b_0} & \frac{c_2 q_2}{\bar{b}_0 - b_0} & q_3 \end{bmatrix} \Rightarrow Q^{-1}A^*Q = D, \ D := \mathrm{diag}(\bar{b}_0, \bar{b}_0, b_0) \tag{5.9b}$$

for all nonzero diagonal entries of Q. This family of transformation matrices does not represent all possible transformation matrices Q with the property $Q^{-1}A^*Q = D$, but for our purposes, we do not need more generality.

Using {(5.9a),(5.9b)}, we can define the transformed iteration method

$$\begin{aligned} (I - D \otimes hJ_{n+1})(\tilde{Y}^{(j)} - \tilde{Y}^{(j-1)}) &= -(Q^{-1} \otimes I)R_n(Y^{(j-1)}), \quad j = 1, \ldots, m, \\ Y^{(j)} &= (Q \otimes I)\tilde{Y}^{(j)}. \end{aligned} \tag{5.10}$$

We shall refer to {(5.9a),(5.9b),(5.10)} as the *transformed* EBDF method. In particular, we may set $c_1 = c_2 = 0$ in (5.9a), i.e. $A^* = D$, so that we can use $Q = I$ which avoids transformation costs. We shall call {(5.8), $A^* = D$} simply the *diagonal* EBDF method. Both iteration processes {(5.9a),(5.9b),(5.10)} and {(5.8), $A^* = D$} have the advantage of possessing a lot of additional intrinsic parallelism when compared with the MEBDF method as implemented in [3], where the three equations in {(5.3a),(5.3c)} are solved sequentially by Newton iteration (to be referred to as *sequential* MEBDF). Firstly, the two LU decompositions can be obtained in parallel; and secondly, in each iteration the forward-backward substitutions for the 3 subsystems and the three components of the residue function $R_n(Y^{(j)})$ can be computed in parallel. Furthermore, in the case of {(5.9a),(5.9b),(5.10)}, the similarity transformation can largely be computed concurrently, at the cost of some data relocation.

Let us compare the iteration cost of diagonal and transformed EBDF on three processors with that of sequential MEBDF. Suppose that respectively m_1, m_2 and m_3 iterations are required to solve {(5.3a),(5.3c)} in sequence. Then, sequential MEBDF requires one LU decomposition, $m_1 + m_2 + m_3$ sequential forward-backward substitutions, and $m_1 + m_2 + m_3$ sequential evaluations of f. Thus, diagonal and transformed EBDF (if we ignore the transformation costs) are less costly than sequential MEBDF if $m < m_1 + m_2 + m_3$. Finally, we remark that in an actual implementation of diagonal and transformed EBDF, it is sometimes advantageous to use in the system matrix in (5.8) the Jacobian J_{n+1} in the blocks of the first and third row and an approximation J_{n+2} of the Jacobian at t_{n+2} in the blocks of the second row (see Section 5.4). Since these Jacobians can again be evaluated concurrently, the effective costs do not increase.

5.3 Convergence of diagonal and transformed EBDF

Let us consider the rate of convergence of the iteration process (5.8). Defining the iteration error $\varepsilon^{(j)} := Y^{(j)} - Y_{n+1}$, we subtract the exact solution relation:

$$(I - A^* \otimes hJ_{n+1})(Y_{n+1} - Y_{n+1}) = -R_n(Y_{n+1})$$

from (5.8) getting

$$\begin{aligned}(I - A^* \otimes hJ_{n+1})(\varepsilon^{(j)} - \varepsilon^{(j-1)}) &= R_n(Y_{n+1}) - R_n(Y_{n+1} + \varepsilon^{(j-1)}) \\ &= -\,\varepsilon^{(j-1)} + h(A \otimes I)\left[F(Y_{n+1} + \varepsilon^{(j-1)}) - F(Y_{n+1})\right] \\ &= -\,(I - A \otimes hJ_{n+1})\varepsilon^{(j-1)} - h(A \otimes I)(I \otimes J_{n+1})\varepsilon^{(j-1)} \\ &\quad + h(A \otimes I)\left[F(Y_{n+1} + \varepsilon^{(j-1)}) - F(Y_{n+1})\right],\end{aligned}$$

or,

$$\begin{aligned}(I - A^* \otimes hJ_{n+1})\,\varepsilon^{(j)} &= (A - A^*) \otimes hJ_{n+1}\varepsilon^{(j-1)} \\ &\quad + h(A \otimes I)\left[F(Y_{n+1} + \varepsilon^{(j-1)}) - F(Y_{n+1}) - (I \otimes J_{n+1})\varepsilon^{(j-1)}\right].\end{aligned}$$

Defining

$$\begin{aligned}&M := (I - A^* \otimes hJ_{n+1})^{-1}\left((A - A^*) \otimes hJ_{n+1}\right), \\ &L := (I - A^* \otimes hJ_{n+1})^{-1}(A \otimes I), \\ &\Phi(\varepsilon) := F(Y_{n+1} + \varepsilon) - F(Y_{n+1}) - (I \otimes J_{n+1})\varepsilon,\end{aligned}$$

the error recursion becomes

$$\varepsilon^{(j)} = M\varepsilon^{(j-1)} + hL\Phi(\varepsilon^{(j-1)}),\ j \geq 1, \tag{5.11}$$

Hence,

$$\varepsilon^{(j)} = M^j\varepsilon^{(0)} + hM^{j-1}L\Phi(\varepsilon^{(0)}) + \cdots + hML\Phi(\varepsilon^{(j-2)}) + hL\Phi(\varepsilon^{(j-1)}). \tag{5.12}$$

5.3.1 The rate of convergence

Let A^* be defined by (5.9a), so that A^* has the same diagonal entries as A. Then the 3-by-3 lower block-triangular matrix M has zero diagonal blocks

$$M = (I - A^* \otimes hJ_{n+1})^{-1} \begin{bmatrix} 0 & & \\ \bar{a}_1\bar{b}_0 hJ_{n+1} & 0 & \\ -c_1 hJ_{n+1} & (b_1 - c_2)hJ_{n+1} & 0 \end{bmatrix}, \tag{5.13}$$

so that M^j vanishes for all $j \geq 3$. Thus,

$$\begin{aligned} \varepsilon^{(1)} &= M\varepsilon^{(0)} + hL\Phi(\varepsilon^{(0)}), \\ \varepsilon^{(2)} &= M^2\varepsilon^{(0)} + hML\Phi(\varepsilon^{(0)})hL\Phi(\varepsilon^{(1)}), \\ \varepsilon^{(j)} &= hM^2L\Phi(\varepsilon^{(j-3)}) + hML\Phi(\varepsilon^{(j-2)}) + hL\Phi(\varepsilon^{(j-1)}), \; j \geq 3. \end{aligned} \tag{5.12$'$}$$

In the case of *linear* problems, where the function Φ vanishes, we have convergence *within three iterations* (of course, if $A^* = A$, then (5.8) reduces to modified Newton, which converges within one iteration for linear problems).

For *nonlinear* problems, we consider the first-order approximation to (5.12′). Let us write $\Phi(\varepsilon) = K\varepsilon + O(\varepsilon^2)$, where K is the (3-by-3 block-diagonal) Jacobian matrix of $\Phi(\varepsilon)$ at $\varepsilon = 0$. Let $N := hLK$, and neglecting terms of degree higher than one in ε, the first-order approximation to (5.12′) becomes

$$\begin{aligned} \varepsilon^{(1)} &= (M + N)\varepsilon^{(0)}, \\ \varepsilon^{(2)} &= (M + N)^2\varepsilon^{(0)}, \\ \varepsilon^{(j)} &= M^2N\varepsilon^{(j-3)} + MN\varepsilon^{(j-2)} + N\varepsilon^{(j-1)}, j \geq 3. \end{aligned} \tag{5.14}$$

In the modified Newton case ($A^* = A$), we have $M = 0$, so that the first-order error recursion (5.14) reduces to $\varepsilon^{(j)} = N\varepsilon^{(j-1)}$, $j \geq 1$. However, if $M \neq 0$, then both N and M play a role in the rate of convergence. We consider the first few iteration errors taking the structure of the matrices M and N into account. From (5.9a) and (5.11) it can be seen that N has a 3-by-3 block lower triangular structure. It follows from the nilpotent structure of M (5.13), and the fact that the product of a lower triangular matrix and a lower triangular nilpotent matrix maintains the same nilpotent structure (independent of the order of multiplication), that all matrix products in (5.14) containing three or more factors M vanish, so that

$$\begin{aligned} \varepsilon^{(1)} &= (M + N)\varepsilon^{(0)}, \\ \varepsilon^{(2)} &= (M^2 + MN + NM + N^2)\varepsilon(0), \\ \varepsilon^{(3)} &= (M^2N + MNM + MN^2 + NM^2 + NMN + N^2M + N^3)\varepsilon^{(0)}, \\ \varepsilon^{(4)} &= (M^2N^2 + (MN)^2 + MN^2M + NM^2N + (NM)^2 + N^2M^2 + O(N^3))\varepsilon^{(0)}, \\ \varepsilon^{(j)} &= O(N^{j-2})\varepsilon^{(0)}, j \geq 5. \end{aligned} \tag{5.14$'$}$$

where the notation $O(N^i)$ is used for terms containing at least i factors N. We summarize the preceding derivations in the following theorem:

Theorem 5.3.1 *In the error recursion (5.11), let the function* Φ *satisfy* $\Phi(\varepsilon) = K\varepsilon + O(\varepsilon^2)$*, and let* $N := hLK$*. Then the first-order approximation to the error recursion (5.12) is given by*

$$\varepsilon^{(j)} = N\varepsilon^{(j-1)}, \ j \geq 1$$

if $A^* = A$ *(modified Newton), or by*

$$\varepsilon^{(j)} = (M+N)^j\varepsilon^{(0)}, \ j = 1,2; \ \varepsilon^{(j)} = O(N^{j-2})\varepsilon^{(0)}, \ j \geq 3$$

if A^* *is defined by (5.9a).*

Thus, if A^* is defined by (5.9a), then after at most two iterations the rate of convergence is comparable with that of modified Newton, for all values of c_1 and c_2. However, in the transformed EBDF case with $c_2 = b_1$, this is already achieved after one iteration, because for this choice M assumes the form (see (5.13))

$$M = \begin{bmatrix} 0 & & \\ \times & 0 & \\ \times & 0 & 0 \end{bmatrix}.$$

Both NM and MN have this same structure; hence, all matrix products in (5.14′) containing two or more factors M vanish. It follows that

$$\varepsilon^{(1)} = (M+N)\varepsilon^{(0)}, \quad \varepsilon^{(j)} = O(N^{j-1})\varepsilon^{(0)}, \ j \geq 2.$$

Thus, we have proved:

Theorem 5.3.2 *Let the conditions of Theorem 5.3.1 be satisfied, and let* $c_2 = b_1$ *in (5.9a). Then, for all* c_1 *the first-order approximation to the error recursion (5.12) associated with transformed EBDF* $\{(5.9a),(5.9b),(5.10)\}$ *is given by*

$$\{\varepsilon^{(1)} = (M+N)\varepsilon^{(0)}; \quad \varepsilon^{(j)} = O(N^{j-1})\varepsilon^{(0)} j \geq 2\}.$$

5.3.2 Amplification factors

Theorems 5.3.1 and 5.3.2 show that transformed and diagonal EBDF may converge slower than Newton in the first iteration and the first two iterations, respectively. The reason is that the magnitude of M is expected to be much greater than that of N. We shall consider the effect of the amplification matrix $(M+N)^j \approx M^j$ on the initial error $\varepsilon^{(0)}$. Although M has only zero eigenvalues, the magnitude of M is not necessarily small. Let us expand $\varepsilon^{(0)}$ with respect to the vectors $a \otimes v$, where v is an eigenvector of the Jacobian matrix J_{n+1}. Since

$$M^j(a \otimes v) = (Z^j(z) \otimes I)(a \otimes v),$$

where $Z(z) := z(I - zA^*)^{-1}(A - A^*)$ and $z := h\lambda(J_{n+1})$, we are interested in the size of $\|Z^j(z)\|_\infty$. It follows from (5.9a) that

$$Z(z) = \frac{z}{(1-b_0z)(1-\bar{b}_0z)} \begin{bmatrix} 0 & 0 & 0 \\ \bar{a}_1\bar{b}_0(1-b_0z) & 0 & 0 \\ (c_1+c_2\bar{a}_1)\bar{b}_0z - c_1 & (b_1-c_2)(1-\bar{b}_0z) & 0 \end{bmatrix}.$$

Assuming that the eigenvalues $\lambda(J_{n+1})$ are in the left halfplane, then $\|Z^j(z)\|_\infty$ is maximal along the imaginary axis, so that we set $z = iy$. Since c_1 and c_2 do not appear in the second row of Z, we can do no better than choosing these parameters such that the third row is inferior to the second in determining the infinity-norm. We verified that this is the case, both for diagonal Newton where $c_1 = c_2 = 0$ and for transformed Newton with $c_1 = 0$ and $c_2 = b_1$. For these cases we find

$$\|Z(iy)\|_\infty = \frac{\bar{a}_1\bar{b}_0|y|}{\sqrt{1+\bar{b}_0^2y^2}}, \ \|Z^2(iy)\|_\infty = \frac{\bar{a}_1\bar{b}_0|b_1-c_2||y^2|}{\sqrt{1+\bar{b}_0^2y^2}\sqrt{1+\bar{b}_0^2y^2}}. \tag{5.15}$$

showing that $\|Z(iy)\|_\infty$ monotonically increases from 0 to $\bar{a}_1$ and $\|Z^2(iy)\|_\infty$ monotonically increases from 0 to $\bar{a}_1|b_1-c_2|\bar{b}_0^{-1}$. Since $\bar{a}_1 > 1$ (see Table 5.2), we should expect that the stiff components in the iteration error are amplified in the first iteration. However, in the second iteration, transformed EBDF already has a zero amplification factor and diagonal EBDF has quite a small amplification factor because $\bar{a}_1|b_1|\bar{b}_0^{-1} \ll 1$. Figure 5.1 illustrates the behavior of $\|Z(iy)\|_\infty$ and $\|Z^2(iy)\|_\infty$ as given by (5.15) for the 6th-order diagonal EBDF method ($k = 5$) used in the numerical experiments.

5.4 Numerical experiments

In this section, we compare the accuracy obtained with a sixth order ($k = 5$) sequential MEBDF method and the transformed ($c_1 = 0$, $c_2 = b_1$) and diagonal ($c_1 = c_2 = 0$) EBDF methods. Our selection of the sixth order method was motivated by the development of a *four*-stage, sixth order *L-stable* method by Psihoyios and Cash [13], the parallelization of which we discuss in a companion article [5] and compare with the three-stage methods of this one. In a full implementation one would of course desire a mechanism for varying the stepsize. However, since at this point we are primarily interested in the algorithmic properties, we will consider only *fixed* stepsizes in order to separate the strategy effects. Some possibilities for variable stepsize implementations use backward difference arrays [3, 7] or Nordsieck vectors [7].

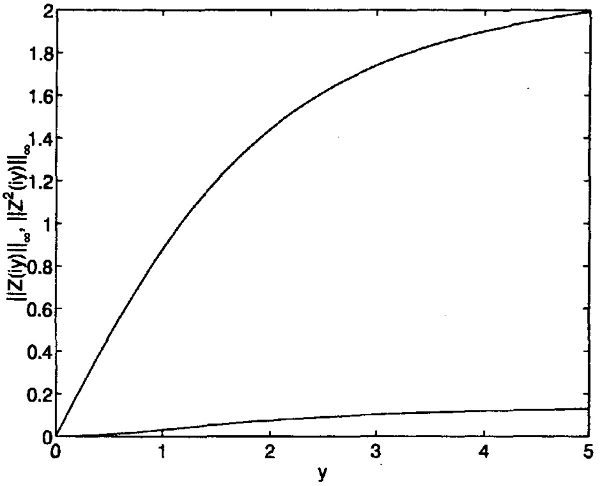

FIGURE 5.1: The amplification factors (5.15) for the 6th-order, iterated EBDF method ($k = 5$).

In all of the experiments, we computed the initial iterates by taking the most recent approximation available at each time level or, if not yet available (in the case of the 'future' value u_{n+2}), by $(k+1)$-point extrapolation of already computed approximations. The experiments include results obtained by the three methods, where the Jacobian matrix J_{n+1} is evaluated in each step using the future-point-approximation to y_{n+1} from the preceding step. Moreover, we included results obtained by diagonal EBDF using two Jacobians J_{n+1} and J_{n+2}, where the y-argument in J_{n+2} is determined by extrapolation of already computed y-values (these two Jacobians can of course be evaluated in parallel). This version will be denoted by EBDF(2). In a realistic implementation, one would only update the Jacobian when necessary to improve convergence.

The starting values were computed either from the exact solution if available or by applying the 5th-order Radau IIA method with a 5 times smaller stepsize. We took two well-known test problems from the literature having no transient phase, which allows us to use fixed stepsizes; viz. a problem posed by Kaps[10].

$$\frac{dy_1}{dt} = -1002y_1 + 1000y_2^2,\ \frac{dy_2}{dt} = y_1 - y_2(1+y_2),\ y_1(0) = y_2(0) = 1,\ 0 \le t \le 5, \tag{5.16}$$

with exact solution $y_1 = e^{-2t}$, $y_2 = e^{-t}$, and the problem

$$\text{HIRES on } [5, 321.8122], \tag{5.17}$$

where the initial conditions at $t = 5$ were obtained by integrating the HIRES problem given in ([8], p. 157) on $[0, 5]$. It turns out that these problems are relatively easy in the sense that the three methods converge within one or two iterations. Therefore, we also used the more difficult problem

$$\begin{array}{lll}
y_1' = -1000(y_1^3 y_2^6 - \cos^3(t)\sin^6(t)) - \sin(t), & & y_1(0) = 1, \\
y_2' = -1000(y_2^5 y_3^4 - \sin^5(t)\sin^4(t)) + \cos(t), & y_2(0) = 0, & 0 \le t \le 1 \\
y_3' = -1000(y_1^2 y_3^3 - \cos^2(t)\sin^3(t)) + \cos(t), & & y_3(0) = 0,
\end{array} \qquad (5.18)$$

with exact solution $y_1 = \cos(t)$ and $y_2 = y_3 = \sin(t)$. Because of its strong nonlinearity it is a more suitable test problem for showing the differences in rate of convergence of the three methods. Finally, we tested the problem

$$\begin{array}{lll}
y_1' = -0.04y_1 + 10^4 y_2 y_3 - 0.96e^{-t}, & & y_1(0) = 1, \\
y_2' = 0.04y_1 - 10^4 y_2 y_3 - 10^7 (y_2)^2 - 0.04e^{-t}, & y_2(0) = 0, & 0 \le t \le t_{\text{end}}, \\
y_3' = 3\,10^7 (y_2)^2 + e^{-t}, & & y_3(0) = 0,
\end{array} \qquad (5.19)$$

with exact solution $y_1 = e^{-t}$, $y_2 = 0$, $y_3 = 1 - e^{-t}$. This problem has the same highly stiff Jacobian matrix as the famous Robertson problem[14], but it is modified by adding nonhomogeneous terms, so that it possesses for the given initial values a solution without transient phase. System (5.19) resembles the original Robertson problem more as t increases. Note that the numerical integration process will become unstable if negative approximations to $y_2(t)$ are generated.

In our numerical experiments, we denoted the number of steps by N, the number of iterations in each iteration process by m , and the total number of iterations by M (not including the iterations needed to compute the starting values). Note that for fixed values of m and N, sequential MEBDF requires three times more ***sequential*** righthand side evaluations and forward-backward substitutions than the transformed and diagonal EBDF-type methods, because sequential MEBDF solves three subsystems per step. Hence, for sequential MEBDF the value of M is three times greater. The accuracy is given by the number of significant correct digits *scd*; that is, we write the maximal ***absolute*** end point error in the form 10^{-scd}. In the tables of results, we shall indicate negative *scd*-values by *.

5.4.1 Fixed numbers of iterations

We start by applying the three methods with a prescribed number of iterations m. In the case of the HIRES problem (5.17) where no exact solution is available, the starting values were provided by the Radau IIA method using 10 iterations. Tables 5.3 and 5.4 list for given values of m and N the resulting *scd*-values for the problems (5.16) and (5.17). These results show that in almost all cases sequential MEBDF finds the solution in one iteration per subsystem, whereas transformed or diagonal EBDF needs two iterations for the whole system (note that transformed and diagonal EBDF show a comparable convergence behavior). Diagonal EBDF(2) behaves poorly for the HIRES problem (5.17) due to the relatively large timesteps which destroy the quality of the Jacobian J_{n+2} (recall that the argument in J_{n+2} is based on extrapolation of preceding y-values). Only for the smallest stepsize in Table 5.4 (i.e. $h \approx 7.9$) does the diagonal EBDF(2) method converge. As to the order behavior, for the Kaps problem the order $p = 6$ of the methods is reproduced, but for the HIRES problem the stepsize is too large to observe asymptotic convergence.

TABLE 5.3: Values of *scd* for problem (5.16).

N	Method	$m = 1$	$m = 2$	$m = 3$	$m = 20$
10	Sequential MEBDF	4.7			4.7
	Transformed EBDF	*	4.5		4.5
	Diagonal EBDF	*	4.7	4.5	4.5
	Diagonal EBDF(2)	*	4.7	4.5	4.5
20	Sequential MEBDF	6.5			6.5
	Transformed EBDF	*	6.3		6.3
	Diagonal EBDF	*	6.4	6.3	6.3
	Diagonal EBDF(2)	*	6.4	6.3	6.3
40	Sequential MEBDF	8.3			8.3
	Transformed EBDF	*	8.1		8.1
	Diagonal EBDF	*	8.2	8.1	8.1
	Diagonal EBDF(2)	*	8.2	8.1	8.1

In order to see more clearly the differences in convergence rates, we now integrate the highly nonlinear problem (5.18). Surprisingly, the numbers of iterations to reach the converged solution is more or less comparable for all methods and differ by at most one iteration. Furthermore, in this example, the additional Jacobian J_{n+2} used in diagonal EBDF(2) improves the initial rate of convergence considerably. The $N = 20$ and $N = 40$ results indicate that again only the stage order $s = 5$ is shown (since the experiments were run with 14 decimals precision, the $N = 80$ results did not reach the expected value $scd = 14.3$). Finally, we integrate the highly stiff modified Robertson problem (5.19) with $t_{\text{end}} = 1$. Here, the performance is similar to that for the Kaps problem (5.16). Apparently, the methods are able to compute positive approximations to the second component $y_2(t)$.

5.4.2 Variable number of iterations

If the number of iterations is adjusted to each nonlinear system (or subsystem in the case of sequential MEBDF) to be solved, then the efficiency is obviously improved,

TABLE 5.4: Values of *scd* for problem (5.17).

N	Method	$m=1$	$m=2$	$m=3$	$m=4$	$m=20$
10	Sequential MEBDF	2.2	2.7	2.8	2.7	2.7
	Transformed EBDF	*	3.1	2.6	2.7	2.7
	Diagonal EBDF	*	2.8	2.5	2.7	2.7
	Diagonal EBDF(2)	*	*	2.4	0.6	*
20	Sequential MEBDF	3.4	3.3			3.3
	Transformed EBDF	*	3.3			3.3
	Diagonal EBDF	*	3.6	3.4	3.3	3.3
	Diagonal EBDF(2)	*	3.2	3.1	3.3	*
40	Sequential MEBDF	4.3	4.2			4.2
	Transformed EBDF	*	4.3			4.3
	Diagonal EBDF	*	4.4	4.3		4.3
	Diagonal EBDF(2)	*	4.3			4.3

TABLE 5.5: Values of *scd* for problem (5.18).

N	Method	$m=1$	$m=2$	$m=3$	$m=4$	$m=5$	$m=6$	$m=20$
20	Seq. MEBDF	5.3	9.3	10.3	10.9	10.8	10.9	10.9
	Transf. EBDF	*	7.6	11.2	11.4	11.3		11.3
	Diag. EBDF	*	5.0	9.8	10.5	10.9	11.4	11.3
	Diag. EBDF(2)	*	11.4	11.3				11.3
40	Seq. MEBDF	11.7	12.3	12.4	12.5	12.4		12.4
	Transf. EBDF	*	12.9	12.8				12.8
	Diag. EBDF	*	11.3	12.3	12.9	12.8		12.8
	Diag. EBDF(2)	*	13.1	12.8				12.8
80	Seq. MEBDF	13.5	13.9	13.8				13.8
	Transf. EBDF	*	13.8					13.8
	Diag. EBDF	*	13.5	13.8				13.8
	Diag. EBDF(2)	*	13.8					13.8

because we avoid the situation where the (sub)system solutions have quite different accuracies. Moreover, in such a dynamic approach, sequential MEBDF can take advantage of the fact that it solves the subsystems *successively* instead of simultaneously as done in the transformed and diagonal EBDF methods. Hence, we also obtain a more

TABLE 5.6: Values of *scd* for problem (5.19) with $t_{\text{end}} = 1$.

N	Method	$m=1$	$m=2$	$m=20$
10	Sequential MEBDF	7.9		7.9
	Transformed EBDF	7.9		7.9
	Diagonal EBDF	7.8	7.9	7.9
	Diagonal EBDF(2)	7.9		7.9
20	Sequential MEBDF	9.6		9.6
	Transformed EBDF	6.4	9.6	9.6
	Diagonal EBDF	*	9.6	9.6
	Diagonal EBDF(2)	4.9	9.6	9.6
40	Sequential MEBDF	11.3		11.3
	Transformed EBDF	*	11.3	11.3
	Diagonal EBDF	*	11.3	11.3
	Diagonal EBDF(2)	*	11.3	11.3

honest comparison.

In our dynamic iteration strategy, we used the stopping strategy described in ([8], p. 130). This stopping strategy depends on a given tolerance parameter *Tol*, because it presupposes the use of automatic stepsize selection based on keeping the local truncation error *LTE* close to *Tol*. Since we focus on convergence aspects we want to use fixed stepsizes, so that we have to replace *Tol* by some estimate of *LTE*. In our case, the difference $u_n - y_n$ from the preceding step provides us with a free estimate of *LTE*. We define the damping parameter θ_m and the accumulated damping parameter η_m:

$$\theta_m := \frac{\|Y^{(m)} - Y^{(m-1)}\|_\infty}{\|Y^{(m-1)} - Y^{(m-2)}\|_\infty}, \quad \eta_0 := (\eta_{\text{old}})^{0.8}, \eta_m := \frac{\theta_m}{1-\theta_m}, m \geq 1, \tag{5.20a}$$

where η_{old} equals the η_m from the preceding step (bounded below by the machine precision). Then, the stopping criterion described in [8] yields for the number of iterations m the condition

$$\eta_m \|Y^{(m)} - Y^{(m-1)}\|_\infty \leq \kappa \|u_n - y_n\|_\infty. \tag{5.21b}$$

Here, κ is a control parameter. The implicit relations are solved more accurately as κ is smaller. For the problems (5.16), (5.17), (5.18) and (5.19), we performed experiments where the number of steps was chosen such that a prescribed *scd*-value was obtained. For these problems, the *maximal* number of iterations in the subsequent iteration processes was prescribed, viz. $m = 5$, $m = 10$, $m = 20$ and $m = 10$, respectively. In problem (5.17), where no exact solution is available, we used the Radau starting method with $m = 10$. Tables 5.7–5.11 list the total number of iterations M needed to obtain a given *scd*-value. Since transformed and diagonal EBDF exhibit a similar convergence behaviour, we only listed *scd*-values for the easier implementable diagonal EBDF methods. From these results we may conclude that the total number of iterations is always less for the diagonal EBDF methods. Furthermore, diagonal EBDF(2) is now performing quite well for the HIRES problem, because the stepsize is adjusted to the required accuracy. On the basis of the above results, we can derive theoretical speedup factors for the efficiency of the iteration part of the methods. Table 5.12 presents such efficiency speedup factors by comparing M-values (averaged over the *scd*-values) for sequential MEBDF and diagonal EBDF(2).

TABLE 5.7: Values of M for problem (5.16) with $\kappa = 0.1$.

Method	$scd = 5$	$scd = 6$	$scd = 7$	$scd = 8$	$scd = 9$	$scd = 10$
Sequential MEBDF	29	49	79	123	187	282
Diagonal EBDF	20	32	59	106	160	244
Diagonal EBDF(2)	20	32	62	97	153	235

TABLE 5.8: Values of M for problem (5.17) with $\kappa = 0.1$.

Method	$scd = 4$	$scd = 5$	$scd = 6$	$scd = 7$
Sequential MEBDF	126	210	305	406
Diagonal EBDF	83	133	189	241
Diagonal EBDF(2)	72	122	177	234

TABLE 5.9: Values of M for problem (5.18) with $\kappa = 0.1$.

Method	$scd = 10$	$scd = 11$	$scd = 12$	$scd = 13$
Sequential MEBDF	103	118	157	231
Diagonal EBDF	131	125	121	140
Diagonal EBDF(2)	32	38	67	105

5.4.3 Code timings

Finally we will give an indication of how our formulation of the diagonal EBDF method compares with the sequential MEBDF method of Cash when implemented on a parallel shared memory machine, in this case a Cray C916. Parallel speedups in this section

TABLE 5.10: Values of M for problem (5.19) with $t_{\mathrm{end}} = 1$ and $\kappa = 0.1$.

Method	$scd = 8$	$scd = 9$	$scd = 10$	$scd = 11$	$scd = 12$	$scd = 13$
Sequential MEBDF	21	39	66	107	168	260
Diagonal EBDF	9	17	29	49	74	114
Diagonal EBDF(2)	8	17	30	48	74	115

TABLE 5.11: Values of M for problem (5.19) with $t_{\mathrm{end}} = 10$ and $\kappa = 0.1$.

Method	$scd = 3$	$scd = 4$	$scd = 5$	$scd = 6$	$scd = 7$	$scd = 8$
Sequential MEBDF	31	60	101	163	255	392
Diagonal EBDF	19	37	47	75	119	184
Diagonal EBDF(2)	18	26	47	76	119	184

TABLE 5.12: Theoretical iteration speedup of diagonal EBDF(2).

Problem	Speedup
(5.16)	1.3
(5.17)	1.7
(5.18)	2.7
(5.19)	2.2

were obtained using the Autotasking Expert analysis tool [4] available on Cray computer systems, which estimates the speedup that would be obtained by a program run on a dedicated multiprocessor system, based on the observed performance on an arbitrarily loaded system. Since the ATExpert tool measures speedup with respect to the same code run on a single processor, it is important for obtaining meaningful results that no redundant work be performed within parallel sections of the code. The tests in this section were run with a *fixed* number of Newton iterations per timestep to clearly distinguish the parallel performance in the absence of iteration strategies. We have taken many more time steps in the experiments of this section to reduce the effects of initialization costs such as memory allocation and startup procedure.

There is, of course, a certain amount of parallelism available in sequential MEBDF. For each of the three relations in {(5.3a),(5.3c)}, a nonlinear system must be solved with a (modified) Newton method, in which the following tasks have varying degrees of parallelism:

1. Evaluation of the Jacobian J_{n+1}.

2. Evaluation of the righthand side.
3. Update of the solution vector.
4. Computation of an LU-decomposition of the system matrix $I - \bar{b}_0 h J_{n+1}$.
5. Execution of a forward-backward substitution.

These tasks all contain a number of independent operations which is proportional to the problem dimension d (*parallelism across the space*, in the classification of Gear[6]) and are present in sequential MEBDF, as well as in the diagonal EBDF methods. However we are interested in an additional, coarser grained parallelism, orthogonal to these parallelizations, such as the concurrent computation of LU-decompositions and forward-backward substitutions for the three subsystems (*parallelism across the method*). This kind of parallelism is not available if the subsystems are solved successively as in sequential MEBDF. However, by solving the subsystems simultaneously as in diagonal EBDF, all of items 1 through 5 above can be computed in parallel for the three subsystems. In the following subsections we present timings concerning the effect of concurrent computation of the various tasks in diagonal EBDF.

LU-decompositions.

Since the computation of LU-decompositions are generally considered to be expensive, we first discuss the effect on the CPU time of computing the LU decomposition of the matrices $I - \bar{b}_0 h J_{n+1}$ and $I - b_0 h J_{n+2}$ needed in diagonal EBDF concurrently. Since the Jacobians are factored only once per time step, the effect of factoring them concurrently becomes less important as more iterations are needed. Table 5.13 shows for $N = 1280$ time steps the speedup figures obtained from a two-processor implementation of diagonal EBDF in which only the two LU-decompositions are computed in parallel. Apparently, for the problems (5.16)–(5.19), the parallel computation of the LU decompositions does not lead to a substantial speedup, even for the 8-dimensional HIRES problem (5.17). Of course, for higher-dimensional problems, the speedup will *increase*. On the other hand, a more sophisticated implementation, where the Jacobian is only updated every few steps, will *decrease* the speedup attained by concurrent decomposition of Jacobians. Therefore, a substantial speedup of a parallel implementation of diagonal EBDF should not be expected from the parallel computation of the LU-decompositions alone.

Overhead costs.

The diagonal EBDF approach incurs a small increase in cost due to the fact that the most recently computed function evaluations $f(u_{n+1}^{(j-1)})$ and $f(u_{n+2}^{(j-1)})$ must be updated in the second and third components of the residue in (5.8), whereas these are constant

TABLE 5.13: Speedups attained by concurrent decomposition of Jacobians in diagonal EBDF.

Problem	$m = 2$	$m = 3$	$m = 4$	$m = 5$
(5.16)	0.97	0.97	1.00	1.00
(5.17)	1.18	1.14	1.11	1.10
(5.18)	1.03	1.02	1.02	1.02
(5.19)	1.04	1.03	1.03	1.02

components of the residue functions if the subsystems are solved in sequence. Hence, these additional costs have to be considered as overhead costs. In order to estimate these costs, we compared diagonal EBDF with sequential EBDF. The latter method is understood to be the method obtained if the EBDF subsystems in {(5.3a),(5.3b)} are solved sequentially. An indication of the significance of this overhead is provided in Table 5.14, in which the ratio of serial CPU times for sequential EBDF and diagonal EBDF is compared for $N = 1280$ time steps. These figures show that the increase in sequential overhead is quite modest.

TABLE 5.14: Ratio of serial CPU times for sequential and diagonal Newton.

Problem	$m = 2$	$m = 3$	$m = 4$	$m = 5$
(5.16)	0.94	0.91	0.89	0.89
(5.17)	0.98	0.97	0.97	0.96
(5.18)	1.01	0.99	0.97	0.97
(5.19)	0.98	0.96	0.95	0.94

Overall speedup factors.

Table 5.15 shows the ATExpert observed speedup of the diagonal EBDF approach on three processors over sequential MEBDF on one processor for $N = 1280$ time steps and $m = 5$ iterations. It is noteworthy that this speedup is essentially independent of the number of Newton iterations. In the table we have also listed the dimension of each system (the nonautonomous terms of problems (5.18) and (5.19) have been implemented as an extra dimension). The attainable speedup is highest for the HIRES problem, which has dimension 8, and lowest for the Kaps problem of dimension 2. As observed in Section 5.4.1, we suffer only a slight loss in convergence rate when changing from sequential MEBDF to diagonal EBDF. Hence, we may expect comparable accuracies for equal numbers of steps N and iterations m, so that the CPU speedup factors in Table 5.15 are also an indication of the speedup of *efficiency* (that is, CPU speedup under the condition of equal accuracies).

TABLE 5.15: Speedup of diagonal EBDF on 3 processors.

Problem	d	$m = 5$
(5.16)	2	1.8
(5.17)	8	2.3
(5.18)	4	2.0
(5.19)	4	2.0

Acknowledgements

The authors are very grateful for the comments and suggestions of the referees.

References

[1] P.N. Brown, A.C. Hindmarsh, and G.D. Byrne. VODE: A variable coefficient ODE solver. Available at `http://www.netlib.org./ode/vode.f`, 1992.

[2] J.R. Cash. On the integration of stiff ODEs using extended backward differentiation formulae. *Numer. Math.*, 34:235–246, 1980.

[3] J.R. Cash. The integration of stiff initial value problems in ODEs using modified extended backward differentiation formulae. *Comput. Math. Appl.*, 5:645–657, 1983. Software available at `http://www.ma.ic.ac.uk/~jcash/IVP_software/finaldae/readme.html`.

[4] Cray Research Inc. *CF77 Commands and directives, SR-3771*, 6.0 edition, 1994.

[5] J.E. Frank and P.J. van der Houwen. Diagonalizable extended backward differentiation formulas. Technical Report MAS-R9917, CWI, Amsterdam, 1999. Submitted for publication.

[6] C.W. Gear. Massive parallelism across time in ODEs. *Appl. Numer. Math.*, 11:27–44, 1993. Proceedings of the International Conference on Parallel Methods for Ordinary Differential Equations, Grado (It), Sept. 10–13, 1991.

[7] E. Hairer, S.P. Norsett, and G. Wanner. *Solving Ordinary Differential Equations, I. Nonstiff Problems.* Springer-Verlag, Berlin, second edition, 1993.

[8] E. Hairer and G. Wanner. *Solving Ordinary Differential Equations, II. Stiff and Differential-Algebraic Problems.* Springer-Verlag, Berlin, second edition, 1996.

[9] E. Hairer and G. Wanner. RADAU. Available at `ftp://ftp.unige.ch/pub/doc/math/stiff/radau.f`, 1998.

[10] P. Kaps. Rosenbrock-type methods. In G. Dahlquist and R. Jeltsch, editors, *Numerical methods for stiff initial value problems, Bericht nr. 9*. Inst. für Geometrie und Praktische Mathematik der RWTH Aachen, 1981.

[11] W.M. Lioen and J.J.B. de Swart. Test set for IVP solvers, Release 2.0. Available at `http://www.cwi.nl/cwi/projects/IVPtestset/`, 1998.

[12] L.R. Petzold. DASSL: A differential/algebraic system solver. Available at `http://www.netlib.org/ode/ddassl.f`, 1991.

[13] G.-Y. Psihoyios and J.R. Cash. A stability result for general linear methods with characteristic function having real poles only. *BIT*, 38:612–617, 1998.

[14] H.H. Robertson. The solution of a set of reaction rate equations. In J. Walsh, editor, *Numerical Analysis, an Introduction*, pages 178–182. Academ. Press, 1966.

Chapter 6

Diagonalizable Extended Backward Differentiation Formulas

Abstract. We generalize the extended backward differentiation formulas (EBDF) introduced by Cash and by Psihoyios and Cash so that the system matrix in the modified Newton process can be block-diagonalized, enabling an efficient parallel implementation. The purpose of this paper is to justify the use of diagonalizable EBDFs on parallel computers and to offer a starting point for the development of a variable stepsize-variable order method. We construct methods which are L-stable up to order $p = 6$ and which have the same computational complexity per processor as the conventional BDF methods. Numerical experiments with the order 6 method show that a speedup factor of between 2 and 4 on four processors can be expected.

6.1 Introduction

In [7] we discussed the parallel implementation of the extended backward differentiation formulas (EBDFs) introduced by Cash in [2] and [3] for the numerical solution of initial value problems for stiff differential equations of the form

$$\frac{d\mathbf{y}}{dt} = \mathbf{f}(t, \mathbf{y}), \ \mathbf{y}, \mathbf{f} \in \mathbb{R}^d, \ t \geq t_0. \tag{6.1}$$

The parallel approach described in [7] is based on block-diagonalization of the system matrix in the modified Newton process used for solving the implicit EBDF relations. The system matrix is of the form $I - (A \otimes hJ)$, where h is the stepsize, I is the identity matrix, the matrix A is determined by the EBDF method coefficients, and J is an approximation to the Jacobian matrix $\partial\mathbf{f}/\partial\mathbf{y}$. Since *exact* block-diagonalization is not possible due to defectiveness of the matrix A, we applied *approximate* block-diagonalization. The resulting block system for the stages can then be efficiently solved in parallel on a number of processors equal to the number of stages. Although the rate of convergence is less than that of true modified Newton, the experiments in [7] show a speedup on a three-processor configuration of between 2 and 3.

The same parallel approach can be applied to the more general EBDF methods which have recently been proposed by Psihoyios and Cash [15]. These more general EBDF methods also lead to defective coefficient matrices A in the modified Newton process, but have the property that they can be made L-stable up to order $p = 6$ (the original

EBDFs are L-stable up to order $p = 4$). However, approximate block diagonalization is now much less accurate than in the case of the original EBDF methods. The aim of this paper is to construct methods which are L-stable up to order $p = 6$ with a nondefective matrix A, so that exact block-diagonalization is possible.

The code MEBDFDAE developed by Cash and Considine [4], is a variable stepsize, variable order implementation of the MEBDF method [3] suitable for integrating differential algebraic equations. This code performed quite well in a comparison with a number of other solvers in [14]. The extension of the results of this paper to differential algebraic equations is the subject of future research.

In Section 6.2, we define a family of EBDF-type methods which generalizes the Cash and Psihoyios-Cash methods. The order conditions, the global error for the Prothero-Robinson test equation, and stability conditions are derived. Section 6.3 discusses the sequential and parallel implementation of these methods and in Section 6.4 we derive L-stable, nondefective EBDF methods of order up to $p = 6$. Per processor, the computational complexity of these methods is comparable to that of the conventional BDF methods. Finally, Section 6.5 reports numerical experiments for the sixth-order method. These experiments indicate that a speedup factor in the range of 2 to 4 on four processors can be expected.

6.2 EBDF-type methods

The generalizations of the EBDF methods to be discussed in this paper are of the form

$$\begin{aligned} &(B \otimes I)\mathbf{Y}_{n+1} - h(C \otimes I)\mathbf{F}(\mathbf{e}t_n + \mathbf{c}h, \mathbf{Y}_{n+1}) = (E \otimes I)\mathbf{V}_n, \\ &\mathbf{V}_n = (\mathbf{y}_{n-s+1}^T, \ldots, \mathbf{y}_n^T)^T. \end{aligned} \tag{6.2}$$

Here, $\otimes$ denotes the Kronecker product, h is the stepsize $t_{n+1} - t_n$, $\mathbf{e}$ and $\mathbf{c}$ are r-dimensional vectors, $\mathbf{e} = (1, \ldots, 1)^T$, $\mathbf{c} = (c_1, \ldots, c_r)^T$ with $c_r = 1$. I is the d by d identity matrix, B and C are r by r lower triangular matrices and E is an r by s matrix. The unknown stage vector $\mathbf{Y}_{n+1}$ contains r stages $\mathbf{y}_{n+c_i}$ of dimension d, representing numerical approximations at the points $t_n + c_i h$, and $\mathbf{F}(\mathbf{e}t_n + \mathbf{c}h, \mathbf{Y}_{n+1})$ contains the r righthand side values $\mathbf{f}(t_n + c_i h, y_{n+c_i})$. Since B and C are lower triangular, the first $r - 1$ stage equations may be considered to be implicit *predictor formulas* providing the internal stage values $\mathbf{y}_{n+c_i}$, $i = 1, \ldots, r-1$, needed in the last stage equation. This last stage equation will be referred to as the *corrector equation* defining the output or step point value $y_{n+c_r} = y_{n+1}$.

We shall call (6.2) an *EBDF-type method*, because it can be viewed as a generalization of the original three-stage EBDF and MEBDF methods of Cash and the four-stage version recently discussed by Psihoyios and Cash. Note that the one-stage versions with $c_1 = 1$ assume the form of the conventional BDF methods.

6.2.1 Nonstiff order of accuracy

Given the abscissa vector $\mathbf{c} = (c_i)$, the matrices B, C and E can be determined such that the ith stage equation in (6.2) is consistent of order p_i provided that $p_i + 1$ free coefficients are available for that equation. To formulate the consistency conditions, we first write (6.1) in autonomous form by adding the equation $dy_{d+1}/dt = 1$, so that (6.2) also becomes autonomous. Next, we introduce the abscissa vector for the back-values $\mathbf{b} := (1-s, 2-s, \ldots, 0)^T$, and the component-wise notation $g(\mathbf{v})$ associated with a scalar function $g : \mathbb{R} \to \mathbb{R}$ to denote the vector with components $g(v_i)$, where $\mathbf{v} = (v_i)$. Upon substitution of the exact solution into (6.2), that is, we set $\mathbf{Y}_{n+1} = \exp(\mathbf{c}h\, d/dt) \otimes \mathbf{y}(t)|_{t_n}$ and $\mathbf{V}_n = \exp(\mathbf{b}h\, d/dt) \otimes \mathbf{y}(t)|_{t_n}$, it is easily seen that the ith stage equation in (6.2) is consistent of order p_i if

$$\mathbf{e}_i^T \left((B - hC) \exp(\mathbf{c}h) - E \exp(\mathbf{b}h) \right) = O(h^{p_i+1}), i = 1, \ldots, r, \tag{6.3}$$

where $\mathbf{e}_i$ is the ith unit vector. The conditions (6.3) lead to the order equations

$$\mathbf{e}_i^T E \mathbf{b}^j = \mathbf{e}_i^T (B\mathbf{c}^j - jC\mathbf{c}^{j-1}),\ j = 0, \ldots, p_i,\ i = 1, \ldots, r, \tag{6.4}$$

where we define $0^0 = 1$. If (6.4) is satisfied, then the *stage order* of (6.2) is defined by $\bar{p} := \min\{p_i\}$. In general, the output value $\mathbf{y}_{n+c_r} = \mathbf{y}_{n+1}$ has *nonstiff* order of accuracy $p = \bar{p}$. However, if the first $r-1$ entries of the last row of the matrix B in (6.2) vanish (as will be the case for the methods of Section 6.4) and if $p_r = \bar{p}+1$ (as will henceforth be assumed), then the nonstiff order of accuracy is equal to $\bar{p} + 1$. The *stiff* order of accuracy is discussed in the following section.

6.2.2 Stiff order of accuracy

We study the global error of the EBDF-type method (6.2) when applied to the Prothero-Robinson equation $dy(t)/dt = \lambda y(t) + \phi(t)$, where ϕ is a given function. By means of this test equation we can obtain insight into the behavior of the error components in the integration of the general ODE system (6.1) by interpreting λ as an eigenvalue of the matrix J, where J denotes the Jacobian of the ODE system. For general linear methods, Hundsdorfer [12] has derived an upper bound for the global error, so that by rewriting (6.2) as a general linear method, we can use his results. However, the rather special form of (6.2) makes it easier to derive such error bounds directly.

Applying (6.2) to the Prothero-Robinson equation yields the EBDF solution

$$y_{n+1} = \mathbf{e}_r^T \mathbf{Y}_{n+1} = \mathbf{e}_r^T (B - zC)^{-1} \left(hC\phi(\mathbf{e}t_n + \mathbf{c}h) + E\mathbf{V}_n \right),\ z := h\lambda. \tag{6.5}$$

Furthermore, upon substitution of the exact solution $y(t)$ into (6.5) we define the local error δ_{n+1} by the relation

$$\begin{aligned} y(t_{n+1}) &= \mathbf{e}_r^T (B - zC)^{-1} \left(hC\phi(\mathbf{e}t_n + \mathbf{c}h) + E\tilde{\mathbf{V}}_n \right) + \delta_{n+1}, \\ \tilde{\mathbf{V}}_n &= \left(y(t_{n-s+1}), \ldots, y(t_n) \right)^T. \end{aligned} \tag{6.6}$$

By subtracting (6.5) from (6.6) and defining the global errors $\varepsilon_n := y(t_n) - y_n$, we obtain

$$\varepsilon_{n+1} = \mathbf{e}_r^T(B - zC)^{-1}E(\varepsilon_{n-s+1}, \ldots, \varepsilon_n)^T + \delta_{n+1}. \tag{6.7}$$

From this global error recursion we derive the following result:

Theorem 6.2.1 *Let $\bar{p}$ be the stage order of the EBDF-type method (6.2). Then, the global error of the Prothero-Robinson equation behaves according to $\varepsilon_{n+1} = O(z^{-1}h^{\bar{p}+1})$ as $h \to 0$ and $z = h\lambda \to \infty$.*

Proof. First an explicit expression for the global error ε_{n+1} in terms of the local errors δ_i is derived. In this derivation, it is convenient to rewrite the multistep difference equation (6.7) in one-step form. Let us define the s-dimensional vector function $\mathbf{u}(z)$, the s-dimensional local error vector $\boldsymbol{\delta}_{n+1}$, and the s-dimensional global error vector $\boldsymbol{\varepsilon}_{n+1}$ by

$$\mathbf{u}^T(z) := \mathbf{e}_r^T(B - zC)^{-1}E, \quad \boldsymbol{\delta}_{n+1} := \delta_{n+1}\mathbf{e}_s, \quad \boldsymbol{\varepsilon}_{n+1} := (\varepsilon_{n-s+2}, \ldots, \varepsilon_{n+1})^T. \tag{6.8}$$

Then, assuming that $\boldsymbol{\varepsilon}_0 = 0$, we obtain

$$\boldsymbol{\varepsilon}_{n+1} = R\boldsymbol{\varepsilon}_n + \boldsymbol{\delta}_{n+1} = \sum_{i=0}^{n} R^i \boldsymbol{\delta}_{n+1-i},$$

$$R = R(z) := \begin{pmatrix} & 1 & & & \\ & & 1 & & \\ & & & \ddots & \\ & & & & 1 \\ u_1(z) & u_2(z) & u_3(z) & \cdots & u_s(z) \end{pmatrix}.$$

Applying partial summation (see [9], p. 242), we arrive at the expression

$$\varepsilon_{n+1} = \mathbf{e}_s^T(I - R)^{-1}(I - R^{n+1})\mathbf{e}_s\delta_1 + \mathbf{e}_s^T\sum_{i=1}^{n}(I - R)^{-1}(I - R^{n+1-i})\mathbf{e}_s(\delta_{i+1} - \delta_i), \tag{6.9}$$

provided that $I - R$ is nonsingular.

Next, we express ε_{n+1} in terms of the derivatives of the exact solution $y(t)$. Using the relation $\phi(t) = y'(t) - \lambda y(t)$, it follows from (6.6) that δ_{n+1} has the Taylor expansion

$$\begin{aligned} &\delta_{n+1} = \sum_{j=0}^{\infty} \frac{1}{j!}\gamma_j(z)h^j y^{(j)}(t_n), \\ &\gamma_0(z) := 1 - \mathbf{e}_r^T(B - zC)^{-1}(E\mathbf{e} - zC\mathbf{e}), \\ &\gamma_j(z) := 1 - \mathbf{e}_r^T(B - zC)^{-1}(E\mathbf{b}^j + jC\mathbf{c}^{j-1} - zC\mathbf{c}^j), \ j \geq 1. \end{aligned} \tag{6.10}$$

From (6.4) we see that $E\mathbf{b}^j = B\mathbf{c}^j - jC\mathbf{c}^{j-1}$ for $j = 0, \ldots, \bar{p}$, $\bar{p}$ being the stage order, so that the first $\bar{p}+1$ terms in the Taylor expansion vanish. Since (6.8) implies $\mathbf{u}^T(z) = -z^{-1}\mathbf{e}_r^T C^{-1}E + O(z^{-2})$, it follows from the structure of the matrix $R(z)$ that $R(z)^n$ vanishes as $z \to \infty$ for $n \geq s$. Hence, we conclude from (6.9) and (6.10) that

$$\begin{aligned}\varepsilon_{n+1} &= \mathbf{e}_s^T(I - R(z))^{-1}\mathbf{e}_s\delta_1(z) + O(h^{\bar{p}+2}) \\ &= \frac{1}{(\bar{p}+1)!}\mathbf{e}_s^T(I - R(z))^{-1}\gamma_{\bar{p}+1}(z)\mathbf{e}_s h^{\bar{p}+1}y^{(\bar{p}+1)}(t_n) + O(h^{\bar{p}+2})\end{aligned}$$

as $z \to \infty$ for $n \geq s$. The theorem now follows from the fact that $(I - R(z))^{-1}$ is bounded as $z \to \infty$ and that $\gamma_{\bar{p}+1}(z) = 1 - \mathbf{e}_r^T\mathbf{c}^{\bar{p}+1} + O(z^{-1}) = O(z^{-1})$. □

If the *stiff order* of accuracy is defined by the order of ε_{n+1} in h as $z \to \infty$, then we conclude from this theorem that the stiff order of EBDF-type methods is $\bar{p}+1$. In [14] the MEBDFDAE code of Cash and Considide [4] is compared with a number of codes based on standard BDF methods—for which the stiff order is equal to the nonstiff order [9], but which are less stable than EBDF-type methods—and Radau IIA methods—for which the nonstiff order of an s stage method is $2s-1$, but for which the stiff order is only $s+1$ [9]. The discussion of this section assumes only the general form (6.2) of EBDF-type methods, and thus applies directly to MEBDF; we think the high accuracy in the presence of stiffness helps to explain the good relative performance of MEBDFDAE observed in [14].

6.2.3 Stability

From the linear difference equation (6.5) it follows that, with respect to the stability test equation $y' = \lambda y$ (i.e. $\phi = 0$), EBDF-type methods are stable if the characteristic equation associated with (6.5) has roots only on the unit disk. Using the identity

$$\mathbf{p}^T P^{-1}\mathbf{q} = \frac{\det(P + \mathbf{q}\mathbf{p}^T)}{\det(P)} - 1,$$

which holds for any nonsingular m by m matrix P and any two m-dimensional vectors $\mathbf{p}$ and $\mathbf{q}$, we find that (6.5) can be written as

$$y_{n+1} = \mathbf{e}_r^T(B - zC)^{-1}E\mathbf{V}_n = \frac{\det(B - zC + E\mathbf{V}_n\mathbf{e}_r^T)}{\det(B - zC)} - 1. \tag{6.5'}$$

Hence, the characteristic equation is given by

$$\zeta^s = \frac{\det(B - zC + E\mathbf{\Gamma}(\zeta)\mathbf{e}_r^T)}{\det(B - zC)} - 1, \ \mathbf{\Gamma}(\zeta) := (1, \zeta, \ldots, \zeta^{s-2}, \zeta^{s-1})^T. \tag{6.11}$$

First of all, we require that (6.2) is zero-stable; that is, we require that for $z = 0$ the characteristic equation (6.11) has one simple root at 1 and $s-1$ roots on the unit disk with only simple roots on the unit circle.

Theorem 6.2.2 *Let B be nonsingular and let the row vectors of the matrix $B^{-1}E$ be denoted by $\mathbf{w}_i^T$. The EBDF-type method (6.2) is zero-stable if the equation $\zeta^s = \mathbf{w}_r^T\mathbf{\Gamma}(\zeta)$ has one simple root $\zeta_1 = 1$ and $s-1$ roots ζ_i, $i = 2, \ldots, s-1$, on the unit disk with only simple roots on the unit circle.*

Proof. For $z = 0$, the characteristic equation (6.11) simplifies to

$$\zeta^s = \frac{\det(B + E\Gamma(\zeta)\mathbf{e}_r^T)}{\det B} - 1 = \det(I + B^{-1}E\Gamma(\zeta)\mathbf{e}_r^T) - 1.$$

The matrix $B^{-1}E\Gamma(\zeta)\mathbf{e}_r^T$ has zero columns, except for its last column which has entries $\mathbf{w}_i^T\mathbf{\Gamma}(\zeta)$, $i = 1, \ldots, r$. Hence, (6.5′) reduces to $\zeta^s = \mathbf{w}_r^T\mathbf{\Gamma}(\zeta)$, which proves the assertion of the theorem. □

Note that this theorem holds for any general linear method of the form (6.2) such that the output (step point) value is given by one of the stages, regardless of the structures of the matrices B, C and E.

Secondly, the stability region of (6.2) is defined by the points in the z-plane where the zeros of (6.11) are on the unit disk. Setting $\zeta = \exp(i\theta)$, the boundary of this region is defined by the boundary locus equation

$$(e^{is\theta} + 1)\det(B - zC) - \det(B - zC + E\Gamma(e^{i\theta})\mathbf{e}_r^T) = 0, \quad 0 \le \theta < 2\pi. \tag{6.12}$$

This equation can be used for plotting stability regions.

Finally, we remark that an A(α)-stable method is automatically L(α)-stable, because the characteristic equation (6.11) reduces to $\zeta^s = 0$ as $z \to \infty$.

6.3 Sequential and parallel iteration

The solution of (6.2) can be obtained by successively solving r subsystems, each of dimension d (recall that B and C are assumed to be lower triangular). If a (modified) Newton method is applied, then the iteration scheme for the ith stage $\mathbf{y}_{n+c_i}$ of $\mathbf{Y}_{n+1}$ assumes the form

$$\begin{aligned}(I - h\tilde{C}_{ii}J)(\mathbf{y}^{(j)}_{n+c_i} - \mathbf{y}^{(j-1)}_{n+c_i}) &= -\mathbf{y}^{(j-1)}_{n+c_i} + h\tilde{C}_{ii}\mathbf{f}(t_{n+c_i}, \mathbf{y}^{(j-1)}_{n+c_i}) \\ &\quad + h\sum_{k=1}^{i-1}\tilde{C}_{ik}\mathbf{f}(t_{n+c_k}, y_{n+c_k}) + \sum_{k=1}^{s}\tilde{E}_{ik}\mathbf{y}_{n-s+k}, \quad j = 1, \ldots, m_i,\end{aligned} \tag{6.13}$$

where $\tilde{C}_{ik}$ and $\tilde{E}_{ik}$ denote the entries of the matrices $B^{-1}C$ and $B^{-1}E$, respectively, J is an approximation to the Jacobian matrix of the righthand side function in (6.1) at t_{n+1}, and $\mathbf{y}^{(0)}_{n+c_i}$ is an initial approximation to $\mathbf{y}_{n+c_i}$. This amounts to the solution of $\bar{m}r$

linear systems per step, where $\bar{m}$ denotes the (average) number of Newton iterations needed in the r subsystems. This approach will be called *sequential* iteration.

If, however, a parallel computer system is available, then one may attempt to solve the r stages more efficiently in parallel on r processors. In [7] we developed for the original EBDF and MEBDF methods of Cash a highly parallel iterative method for solving the implicit relations in (6.2). This parallel approach can also be applied to methods of the form (6.2) with more general matrices B, C and E. It is based on the approximate block-diagonalization of the modified Newton method applied to the full (block) system (6.2). Let us define the residue function

$$\mathbf{R}_n(\mathbf{Y}) := \mathbf{Y} - h(B^{-1}C \otimes I)\mathbf{F}(\mathbf{e}t_n + \mathbf{c}h, \mathbf{Y}) - (B^{-1}E \otimes I)\mathbf{V}_n. \tag{6.14}$$

Then, solving (6.2) by m modified Newton iterations amounts to

$$(I - B^{-1}C \otimes hJ)(\mathbf{Y}_{n+1}^{(j)} - \mathbf{Y}_{n+1}^{(j-1)}) = -\mathbf{R}_n(\mathbf{Y}_{n+1}^{(j-1)}),\ j = 1, \ldots, m. \tag{6.15}$$

The MEBDF methods were developed in [3] to avoid factoring two Jacobians by forcing the diagonal of $B^{-1}C$ to be a constant. In parallel the extra factorizations come for free, and a constant diagonal is actually undesirable: if we use an abscissa vector of the form $\mathbf{c} = (1, 2, \ldots, r-1, 1)^T$ and assume the same zero structure of the matrices B, C and E as in the original EBDF and MEBDF methods, then the matrix $B^{-1}C$ is defective, so that we cannot directly diagonalize (6.13) by applying a similarity transformation. One option is to replace the matrix $B^{-1}C$ in (6.15) by a diagonalizable approximation A^*, for example, by $A^* = \mathrm{diag}(B^{-1}C)$. The rate of convergence will be less than that of the modified Newton method, however. In the case of the three-stage EBDF and MEBDF methods, the loss in rate of convergence is modest (see the experiments in [7]) because the diagonalizable approximation is quite accurate. In fact, even with the simple choice $A^* = \mathrm{diag}(B^{-1}C)$, we obtained surprisingly fast convergence. However, for higher-stage methods, where diagonalizable approximations are less accurate, the rate of convergence is expected to decrease significantly.

6.3.1 Nondefective methods

Rather than applying approximate block-diagonalization, we follow an alternative approach in which the abscissa vector is changed to the form $\mathbf{c} = (c_1, 2, 3, \ldots, r-1, 1)^T$ and in which we choose $c_1 \neq 1$ such that $B^{-1}C$ is no longer a defective matrix (except for the degenerate case $s < r - 2$). We shall call such EBDF methods *nondefective EBDF methods.* Nondefective EBDF methods can directly be diagonalized by the transformation $\tilde{\mathbf{Y}}^{(j)} = (Q^{-1} \otimes I)\mathbf{Y}^{(j)}$, where Q is such that $Q^{-1}(B^{-1}C)Q = D$ with D diagonal. This yields the transformed iteration method

$$\begin{aligned}(I - D \otimes hJ)(\tilde{\mathbf{Y}}^{(j)} - \tilde{\mathbf{Y}}^{(j-1)}) &= -(Q^{-1} \otimes I)\mathbf{R}_n\left((Q \otimes I)\tilde{\mathbf{Y}}^{(j-1)}\right),\ j = 1, \ldots, m,\\ \mathbf{Y}_{n+1} &= (Q \otimes I)\tilde{\mathbf{Y}}^{(m)}\end{aligned} \tag{6.16}$$

and will be called *parallel* iteration. We emphasize that (6.16) is algebraically equivalent to (6.15).

The introduction of the free parameter c_1 in the abscissa vector $\mathbf{c} = (c_1, 2, \ldots, r-1, 1)^T$ preserves the attractive property that all stage values, except for the first one, can be reused in the initial approximation $\mathbf{Y}^{(0)}$ needed in the succeeding time step. The first stage can in turn be approximated by interpolating between the abundant history and stage values. In fact, setting $c_1 = 1$ has no additional advantages, because it 'duplicates' the output value at t_{n+1}.

6.3.2 Convergence condition

Defining the iteration error $\varepsilon^{(j)} := \mathbf{Y}^{(j)} - \mathbf{Y}_{n+1}$, we derive for (6.15) the error recursion

$$\begin{aligned}
\varepsilon^{(j)} &= hK\Phi(\varepsilon^{(j-1)}), \; j = 1, \ldots, m, \\
K &:= (I - B^{-1}C \otimes hJ)^{-1}(B^{-1}C \otimes I), \\
\Phi(\varepsilon) &:= \mathbf{F}(\mathbf{e}t_n + \mathbf{c}h, \mathbf{Y}_{n+1} + \varepsilon) - \mathbf{F}(\mathbf{e}t_n + \mathbf{c}h, \mathbf{Y}_{n+1}) - (I \otimes J)\varepsilon.
\end{aligned}$$

Let $\Phi(\varepsilon)$ have at $\varepsilon = 0$ a Lipschitz constant L_Φ with respect to the Euclidean norm and let the problem be dissipative, i.e. $\mu_2[J] \le 0$, where $\mu_2[\cdot]$ denotes the logarithmic norm associated with the Euclidean norm. Then, by applying the matrix version of von Neumann's theorem (see [9], p. 356), we conclude that for dissipative problems

$$\|\varepsilon^{(j)}\|_2 \le hL_\Phi L_K \|\varepsilon^{(j-1)}\|_2, \; L_K = \max\{\|(I - zB^{-1}C)^{-1}B^{-1}C\|_2 : \mathrm{Re}(z) \le 0\}. \tag{6.17}$$

Hence, for dissipative problems, a sufficient condition for convergence is

$$h \le \frac{1}{L_\Phi L_K}.$$

Thus, difference in convergence of two EBDF-type methods is mainly determined by differences in the upper bound L_K.

6.3.3 Analysis of computational expenses

Finally, the computational expenses of (6.13) when implemented on one processor (sequential iteration of the subsystems) are compared with those of (6.16) implemented on r processors (parallel iteration). In (6.13) we define $\bar{m} := r^{-1}(m_1 + \cdots + m_r)$ and we denote the number of distinct diagonal entries of C by r_0. Table 6.1 lists the numbers of floating point operations to advance the solution one time step using a fixed stepsize. In this table, C_f and C_J respectively denote the average numbers of operations needed to compute a component of f and an entry of J.

TABLE 6.1: Operation costs per processor to advance the solution one time step.

	Sequential iteration	Parallel iteration
Once per Jacobian update		
Jacobian evaluation	$C_J d^2$	$\frac{1}{r} C_J d^2$
System matrix	$r_0 d$	d
LUD of system matrix	$\frac{2}{3} r_0 d^3$	$\frac{2}{3} d^3$
Once per time step		
Righthand side	$(C_f + 2s + 1)rd - (C_f + 2)d$	$(2s - 1)d$
Per Newton iteration		
Forward/backward	$2rd^2$	$2d^2$
Updates	$r(C_f + 5)d$	$(C_f + r + 4)d$
Transformations	–	$2rd$

For a linear problem, only one Newton iteration is needed. Hence, assuming that the costs of building and factoring the Jacobian are negligible, it follows from Table 6.1 that the parallel speedup can be estimated by

$$S = r \frac{(2 - r^{-1})C_f + 2d + 2s + 6 - 2r^{-1}}{C_f + 2d + 2s + 3r + 3}.$$

At the other extreme, assume a very stiff nonlinear problem such that the Jacobian must be evaluated once per step. Then, we obtain

$$S = r \frac{\bar{m}(2d + C_f + 5) + (C_f + 2s + 1) + r^{-1}(C_J d - C_f - 2) + r_0 r^{-1}(1 + \frac{2}{3}d^2)}{m(2d + C_f + 3r + 4) + (2s - 1) + r^{-1} C_J d + (1 + \frac{2}{3}d^2)},$$

from which the following observations can be made:

- If the evaluation of the Jacobian dominates the computation, then $S \approx r$.
- If factoring the Jacobian dominates the computation, then $S \approx r_0$.
- If the iterations dominate the computation, then $S \approx r\bar{m}m^{-1}$.

6.4 Construction of nondefective EBDF methods

We shall construct nondefective versions of the original three-stage and four-stage EBDF-type methods given in [2] and [15].

6.4.1 Three-stage methods

We consider methods of the form (6.2) with $r = 3$ and

$$c = \begin{pmatrix} c_1 \\ 2 \\ 1 \end{pmatrix}, \quad B = \begin{pmatrix} 1 & 0 & 0 \\ B_{21} & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad C = \begin{pmatrix} C_{11} & 0 & 0 \\ 0 & C_{22} & 0 \\ C_{31} & C_{32} & C_{33} \end{pmatrix},$$
$$E = \begin{pmatrix} E_{11} & E_{12} & \cdots & E_{1s} \\ 0 & E_{22} & \cdots & E_{2s} \\ E_{31} & E_{32} & \cdots & E_{3s} \end{pmatrix}. \tag{6.18}$$

Given the abscissa c_1 and one of the parameters C_{3j}, the remaining entries in the arrays in (6.18) can be computed by means of the order conditions (6.4) such that $p_1 = p_2 = s$ and $p_3 = s + 1$. Hence, the order of accuracy (both stiff and nonstiff) is $p = s + 1$. The cases $\{c_1 = 1, C_{31} = 0\}$ and $\{c_1 = 1, C_{33} = C_{11} = C_{22}\}$ respectively define the original EBDF and MEBDF methods. For future reference, Table 6.2 lists for $p = 3, \ldots, 6$ the MEBDF values of the angle of unconditional stability α; the parameters D_1 and D_2 determining the rectangle $\{z : -D_1 \leq \Re(z) \leq 0, -D_2 \leq \Im(z) \leq D_2\}$ containing the region of instability in the left half-plane; and the maximal modulus of the characteristic roots ζ in this region of instability. For larger values of p, the angle α quickly decreases, so that the resulting integration methods are less useful for solving general stiff problems.

TABLE 6.2: Three-stage MEBDF methods of Cash with $c_1 = 1$, $C_{33} = C_{11} = C_{22}$.

p	3	4	5	6
α	90°	90°	88.4°	83.1°
(D_1, D_2)	(0,0)	(0,0)	(0.040, 1.8)	(0.246, 2.6)
$\lvert\zeta\rvert_{\max}$	1	1	1.029	1.121

As we already observed, the MEBDF methods of Table 6.2 are defective, so that direct diagonalization is not possible. Therefore, we used the two free parameters c_1 and C_{31} to construct a nondefective, zero-stable and L(α)-stable EBDF method with (i) a relatively large α and (ii) a well-conditioned transformation matrix Q. Requiring that Q be lower triangular with unit diagonal entries, we found by a straightforward numerical search the results listed in Table 6.3 (for the L-stable third- and fourth-order methods, the generating matrices $B^{-1}C$, $B^{-1}E$, D and Q needed in (6.16) are given in the Appendix to this paper). We mention only that there is a lot of freedom in choosing the parameters c_1 and C_{31} to determine L-stable 3-stage methods satisfying (i) and (ii). For the 4-stage methods of the next section, the L-stable parameter space is much more restricted.

TABLE 6.3: Three-stage, nondefective EBDF methods of the form (6.18).

p	3	4	5	6
c_1	5/4	5/4	5/4	5/4
C_{31}	0	0	2/7	3/13
$\|Q\|_\infty$	6.1	7.9	6.6	8.3
α	90°	90°	88.5°	83.9°
(D_1, D_2)	(0, 0)	(0, 0)	(0.04, 2.1)	(0.24, 3.9)
$\lvert\zeta\rvert_{\max}$	1	1	1.029	1.121

6.4.2 Higher-stage methods

The original EBDF and MEBDF methods have $c = (1,2,1)^T$, so that there is one 'future point' at $t_n + 2h$. This method can be interpreted as the successive application of the s-step BDF formula at $t_n + h$ and $t_n + 2h$ for predicting the future point value at $t_n + 2h$ needed in the $(s+1)$-step (M)EBDF corrector formula. More generally, we may introduce further future points by using $c = (1,2,3,\dots,r-1,1)^T$. Considering only the stability of the corrector formula (last stage equation), we verified experimentally that up to order 18 the maximal order of L-stable formulas increases by 2 and the maximal order of L(α)-stable formulas increases by 3 with each additional future point. Of course, the use of BDF predictors will reduce the stability of the overall method, but we may still hope for improvement: Psihoyios and Cash [15] have shown that there exist L-stable 4-stage methods of order 6. However, just as in the case of the three-stage EBDF, choosing $c = (1,2,3,\dots,r-1,1)^T$ yields defective matrices $B^{-1}C$. Therefore, we shall consider abscissae vectors of the form

$$c = (c_1, 2, 3, \dots, r-1, 1)^T, \tag{6.19a}$$

where c_1 is a free parameter. According to the structure of the original (M)EBDF methods, we impose the following sparsity pattern on the matrices B, C and E:

$$B := \begin{pmatrix} 1 & & & \\ \vdots & \ddots & & \\ * & \cdots & 1 & \\ 0 & \cdots & 0 & 1 \end{pmatrix}, \quad C := \begin{pmatrix} * & & & \\ & \ddots & & \\ & & * & \\ * & \cdots & * & * \end{pmatrix}, \quad E := \begin{pmatrix} * & \cdots & * & \cdots & * \\ & & \ddots & & \vdots \\ & & * & \cdots & * \\ * & \cdots & * & \cdots & * \end{pmatrix}, \tag{6.19b}$$

The entries in the matrices B, C and E can be determined such that the first $r-1$ stage equations in (6.19b) are consistent of order s. The last stage equation contains $r+s$ free parameters, so that it can be made consistent of order $r+s-1$. Since the order of accuracy of (6.19a,6.19b) cannot exceed $s+1$, we shall choose the entries in the corrector equation such that it is consistent of order $s+1$, leaving $r-2$ free parameters. Together with the free parameter c_1, we obtain an $(r-1)$-parameter family of EBDF-type methods with stage order $\bar{p} = s$ and order of accuracy $p = s+1$.

From this family, we want nondefective, L-stable methods, again under the condition of zero-stability and a well-conditioned transformation matrix Q.

Let us consider the case of four stages ($r = 4$) with three free parameters. As already mentioned, Psihoyios and Cash have considered the defective case $c_1 = 1$ and shown that L-stable, sixth-order methods exist for a particular choice of the remaining two free parameters. For example, they verified that the parameters $C_{41} = 1/10$ and $C_{43} = 1/20$ generate an L-stable method with $p = s + 1 = 6$. This motivated us to search for nondefective, L- and zero-stable methods by choosing $c_1 \neq 1$. A numerical search produced for $p = s + 1 = 5$ the values $c_1 = 3/2$, $C_{41} = 3/10$, $C_{43} = 7/50$ giving $\|Q\|_\infty \approx 31.5$ and for $p = s + 1 = 6$ the values $c_1 = 6/5$, $C_{41} = 11/100$, $C_{43} = 1/20$, giving $\|Q\|_\infty \approx 167.5$. The corresponding generating matrices $B^{-1}C$, $B^{-1}E$, D and Q needed in (6.16) are given in the Appendix.

Together with the conventional BDF methods of order $p = 1$ and $p = 2$, and the three-stage nondefective EBDF methods of order $p = 3$ and $p = 4$ derived in the preceding section, we now have L-stable methods up to order six, all having a comparable effective computational complexity per step, provided that we employ three processors for $p = 3, 4$ and four processors for $p = 5, 6$.

6.5 Numerical experiments

Preliminary numerical experiments have been conducted using a constant stepsize implementation in Matlab. Due to the difficulty of determining the free coefficients for optimal stability, a variable stepsize implementation should be based on interpolation of back-values to maintained evenly-spaced data. One could also recompute the coeffients at each timestep for fixed values of the free parameters, but this is likely to lead to a loss of L-stability at high order. In fact, this paper is meant as a starting point for the development of a variable stepsize-variable order code.

In the numerical experiments we compare two methods from the three-parameter family of four-stage, 6th-order EBDF-type methods of the form (6.19a,6.19b) with free parameters c_1, C_{41} and C_{43}. The first method is due to Psihoyios and Cash and is defined by $c_1 = 1$, $C_{41} = 0.10$, $C_{43} = 0.05$. It is L-stable, but defective, so that sequential iteration has to be applied (see Section 2.4). The second method is defined by $c_1 = 1.2$, $C_{41} = 0.11$, $C_{43} = 0.05$. It also is L-stable, but nondefective, so that the parallel iteration method (6.16) can be applied. In the following, we call these methods the Defective and Nondefective EBDF methods, respectively. The values of the parameter L_K in (6.17) are $L_K \approx 1.88$ for the Defective EBDF method and $L_K \approx 1.68$ for the Nondefective EBDF method, so we would expect the methods to have similar convergence behaviors. In addition to these methods, we reproduced the results from [7] obtained for the original three-stage, 6th-order EBDF method of Cash when iterated by the diagonal iteration method (6.14) with $A^* = \mathrm{diag}(B^{-1}C)$, to be referred to as Diagonal EBDF. By mutual comparison of the three methods we can see what we have gained by the introduction of nondefective EBDF methods.

Following [7] the initial iterates for the iteration processes are obtained by taking the most recent approximation available or, if not yet available (in the case of the future value at t_{n+r-1} and at t_{n+c_1}), by 6-point extrapolation of already computed approximations. The Jacobian matrix J is evaluated in each step using the future-point-approximation to y_{n+1} from the preceding step. The starting values were obtained either from the exact solution (if available) or by applying the 5th-order Radau IIA method with a 5 times smaller stepsize and using 10 Newton iterations per step.

Three of the test problems are the same as in [7], viz. the Kaps problem [13]

$$\begin{aligned}&\frac{dy_1}{dt} = -1002y_1 + 1000y_2^2,\\&\frac{dy_2}{dt} = y_1 - y_2(1+y_2),\\&y_1(0) = y_2(0) = 1, 0 \le t \le 5;\end{aligned} \tag{6.20}$$

the eight-dimensional 'High Irradiance RESponse' problem given in ([9], p. 157):

$$\text{HIRES on the interval } [5, 321.8122], \tag{6.21}$$

where the initial conditions at $t = 5$ were obtained by applying the RADAU 5 code [10] on $[0,5]$; and the non-autonomous Robertson problem, modified to remove the transient phase and make the problem suitable for fixed stepsize integration:

$$\begin{array}{lll} y_1' = -0.04y_1 + 10^4 y_2 y_3 - 0.96\mathrm{e}^{-t}, & y_1(0) = 1, & \\ y_2' = 0.04y_1 - 10^4 y_2 y_3 - 10^7 (y_2)^2 - 0.04\mathrm{e}^{-t}, & y_2(0) = 0, & 0 \le t \le 1. \\ y_3' = 3 \times 10^7 (y_2)^2 + \mathrm{e}^{-t}, & y_3(0) = 0, & \end{array} \tag{6.22}$$

The fourth test problem is the 15-dimensional circuit analysis problem due to Horneber [11] and extensively discussed in [6] and [8]. In our implementation, we used the specification given in [14]:

$$\text{Ring modulator on the interval } [0, 10^{-3}] \text{ with } C_s = 10^{-9}. \tag{6.23}$$

In our numerical experiments, we denoted the number of steps by N, the number of iterations in each iteration process by m, and the total number of iterations by M (not including the iterations needed to compute the starting values). Note that for fixed values of m and N, a serial implementation of Defective EBDF requires—per processor—four times as many righthand side evaluations and forward-backward substitutions as a 4-processor parallel implementation of the Nondefective EBDF method, because Defective EBDF solves four subsystems per step. Hence, we would expect the value of M to be four times greater for Defective EBDF. The accuracy is given by the number of significant correct digits *scd*; that is, we write the maximal *absolute* end point error in the form 10^{-scd}. In the tables of results, we shall indicate negative *scd*-values by *.

As a basis for comparison of parallel performance, we list here the rough speedup results obtained by two other parallel ODE solvers. Bendtsen [1] reports speedups of between 3 and 5 on a 9-processor implementation of an eighth order multiple implicit Runge-Kutta methods. In [5], de Swart reports speedups of between 2 and 4 from a 4-processor implementation of the fifth order Radau IIA method.

6.5.1 Fixed numbers of iterations

The Tables 6.4, 6.5 and 6.6 list for given values of m and N the resulting *scd*-values for the first three problems (6.20)–(6.22). These results show that the three methods converge to solutions with comparable accuracy. Furthermore, the convergence rate is for Diagonal EBDF slightly less than for the other two methods.

TABLE 6.4: Values of *scd* for the Kaps problem (6.20).

N	Method	$m=1$	$m=2$	...	$m=\infty$
10	Defective EBDF	5.0		...	5.0
	Nondefective EBDF	5.2		...	5.2
	Diagonal EBDF	*	4.7	...	4.5
20	Defective EBDF	6.8		...	6.8
	Nondefective EBDF	6.9		...	6.9
	Diagonal EBDF	*	6.4	...	6.3
40	Defective EBDF	8.5		...	8.5
	Nondefective EBDF	8.8		...	8.8
	Diagonal EBDF	*	8.2	...	8.1

TABLE 6.5: Values of *scd* for the HIRES problem (6.21).

N	Method	$m=1$	$m=2$	$m=3$	$m=4$	...	$m=\infty$
10	Defective EBDF	3.1	3.0	2.0	3.2	...	3.1
	Nondefective EBDF	3.4	3.0	2.0	2.9	...	2.8
	Diagonal EBDF	*	2.8	2.5	2.7	...	2.7
20	Defective EBDF	2.6	3.7	3.9	3.9	...	3.8
	Nondefective EBDF	3.6	3.7	3.6	3.6	...	3.6
	Diagonal EBDF	*	3.6	3.4	3.3	...	3.3
40	Defective EBDF	5.0	4.8	4.9		...	4.9
	Nondefective EBDF	4.4	4.7	4.8		...	4.8
	Diagonal EBDF	*	4.4	4.3		...	4.3

TABLE 6.6: Values of *scd* for the Robertson problem (6.22).

N	Method	$m=1$	$m=2$	...	$m=\infty$
10	Defective EBDF	7.6		...	7.6
	Nondefective EBDF	7.7		...	7.7
	Diagonal EBDF	7.8	7.9	...	7.9
20	Defective EBDF	9.3		...	9.3
	Nondefective EBDF	9.3		...	9.3
	Diagonal EBDF	*	9.6	...	9.6
40	Defective EBDF	11.0		...	11.0
	Nondefective EBDF	11.0		...	11.0
	Diagonal EBDF	*	11.3	...	11.3

6.5.2 Variable number of iterations

For our dynamic iteration strategy, we used the stopping criterion described in ([9], p. 130), replacing the tolerance parameter *Tol* with an estimate of the local truncation error *LTE* (see [7] for details on this modification). At the timestep $(n+1)$, *LTE* was defined to be the difference between the order p approximation to $\mathbf{y}_n$ from the last stage of $\mathbf{Y}_n$ and—in the case of Defective EBDF—the order $p-1$ approximation from the first stage of $\mathbf{Y}_n$ or—in the Nondefective EBDF case—the order $p-1$ approximation from the second stage of $\mathbf{Y}_{n-1}$. All further iteration strategy parameters are the same as in [7].

For the three most difficult problems (6.21), (6.22) and (6.23), we performed experiments in which the number of steps was chosen such that a prescribed *scd*-value was obtained. For these problems, the *maximal* number of Newton iterations in the subsequent iteration processes was limited to 10. The Tables 6.7, 6.8 and 6.9 list the total number of iterations M needed to obtain a given *scd*-value. From these values, we may conclude that the two parallel methods Nondefective EBDF and Diagonal EBDF need about two to four times fewer iterations then the sequential method Defective EBDF.

TABLE 6.7: Values of M for the HIRES problem (6.21).

Method	*scd* = 4	*scd* = 5	*scd* = 6	*scd* = 7
Defective EBDF	96	168	573	928
Nondefective EBDF	73	102	195	343
Diagonal EBDF	83	133	189	241

Acknowledgements

The authors are grateful to Jeff Cash for his interest and his comments on this work, and for drawing our attention to [15].

TABLE 6.8: Values of M for the Robertson problem (6.22).

Method	$scd = 8$	$scd = 9$	$scd = 10$	$scd = 11$	$scd = 12$	$scd = 13$
Defective EBDF	36	63	107	169	265	415
Nondefective EBDF	10	18	29	47	74	123
Diagonal EBDF	9	17	29	49	74	114

TABLE 6.9: Values of M for the Ring modulator (6.23).

Method	$scd = 6$	$scd = 7$	$scd = 8$
Defective EBDF	49900	72400	104500
Nondefective EBDF	14800	22300	33800
Diagonal EBDF	20000	29700	42800

Appendix: Coefficients of some nondefective EBDF methods

For reference we provide the coefficient matrices of the L-stable nondefective EBDF methods considered in this paper. For each method we give the matrices $B^{-1}C$, $B^{-1}E$ and Q needed for parallel implementation of (6.16). Obviously, the diagonal matrix D needed for the implementation of 6.16 is given by $D = \operatorname{diag}(B^{-1}C)$. The coefficients listed are exact, expressed in fractional form, and were determined by Maple.

The 3-stage L-stable method of order $p = 3$ is defined by $c_1 = 5/4$, $C_{31} = 0$. The method coefficients and transformation matrix are given by

$$B^{-1}C = \begin{pmatrix} \frac{45}{56} & 0 & 0 \\ \frac{72}{77} & \frac{6}{11} & 0 \\ 0 & -\frac{4}{23} & \frac{22}{23} \end{pmatrix}, \quad B^{-1}E = \begin{pmatrix} -\frac{25}{56} & \frac{81}{56} \\ -\frac{40}{77} & \frac{117}{77} \\ -\frac{5}{23} & \frac{28}{23} \end{pmatrix}, \tag{6.24}$$
$$Q = \begin{pmatrix} 1 & 0 & 0 \\ \frac{192}{53} & 1 & 0 \\ \frac{43008}{10441} & \frac{11}{26} & 1 \end{pmatrix}.$$

A 3-stage L-stable method of order $p = 4$ is defined by $c_1 = 5/4$, $C_{31} = 0$. The method coefficients and transformation matrix are given by

$$
\begin{gathered}
B^{-1}C = \begin{pmatrix} \frac{585}{908} & 0 & 0 \\ \frac{192}{227} & \frac{6}{13} & 0 \\ 0 & -\frac{18}{197} & \frac{150}{197} \end{pmatrix}, \quad B^{-1}E = \begin{pmatrix} \frac{2025}{7264} & -\frac{4225}{3632} & \frac{13689}{7264} \\ \frac{1080}{2951} & -\frac{4204}{2951} & \frac{6075}{2951} \\ \frac{17}{197} & -\frac{99}{197} & \frac{279}{197} \end{pmatrix}, \\
Q = \begin{pmatrix} 1 & 0 & 0 \\ \frac{3328}{719} & 1 & 0 \\ \frac{18130944}{5022215} & \frac{39}{128} & 1 \end{pmatrix}.
\end{gathered}
\tag{6.25}
$$

A 4-stage L-stable method of order $p = 5$ is defined by $c_1 = 3/2$, $C_{41} = 3/10$, $C_{43} = 7/50$. The method coefficients and transformation matrix are given by

$$
B^{-1}C = \begin{pmatrix} \frac{315}{496} & 0 & 0 & 0 \\ \frac{864}{1147} & \frac{12}{37} & 0 & 0 \\ \frac{2768}{3441} & \frac{32}{37} & \frac{4}{9} & 0 \\ \frac{3}{10} & -\frac{3059487}{4001600} & \frac{7}{50} & \frac{5279163}{4001600} \end{pmatrix},
$$

$$
B^{-1}E = \begin{pmatrix} -\frac{1225}{3968} & \frac{6075}{3968} & -\frac{11907}{3968} & \frac{11025}{3968} \\ -\frac{420}{1147} & \frac{2043}{1147} & -\frac{3884}{1147} & \frac{3408}{1147} \\ -\frac{12110}{30969} & \frac{2118}{1147} & -\frac{3907}{1147} & \frac{91382}{30969} \\ \frac{2153579}{24009600} & -\frac{3413921}{8003200} & \frac{4631823}{8003200} & \frac{3640463}{4801920} \end{pmatrix}, \tag{6.26}
$$

$$Q = \begin{pmatrix} 1 & 0 & 0 & 0 \\ \frac{4608}{1901} & 1 & 0 & 0 \\ \frac{24616704}{1617751} & -\frac{36}{5} & 1 & 0 \\ -\frac{38599642812960}{45767552496101} & \frac{145802607}{81838795} & -\frac{5042016}{31506067} & 1 \end{pmatrix}.$$

A 4-stage L-stable method of order $p = 6$ is defined by $c_1 = 6/5$, $C_{41} = 11/100$, $C_{43} = 1/20$. The method coefficients and transformation matrix are given by

$$B^{-1}C = \begin{pmatrix} \frac{16016}{32525} & 0 & 0 & 0 \\ \frac{40625}{49438} & \frac{15}{38} & 0 & 0 \\ \frac{39040625}{41626796} & \frac{30375}{31996} & \frac{180}{421} & 0 \\ \frac{11}{100} & -\frac{120153318}{388515625} & \frac{1}{20} & \frac{1497086157}{1554062500} \end{pmatrix},$$

$$B^{-1}E = \begin{pmatrix} \frac{569184}{4065625} & -\frac{10469888}{12196875} & \frac{9018009}{4065625} & -\frac{12719616}{4065625} & \frac{32064032}{12196875} \\ \frac{5775}{24719} & -\frac{101768}{74157} & \frac{82350}{24719} & -\frac{105400}{24719} & \frac{227750}{74157} \\ \frac{5549775}{20813398} & -\frac{46526500}{31220097} & \frac{70906923}{20813398} & -\frac{42611025}{10406699} & \frac{90894625}{31220097} \\ -\frac{211339877}{6216250000} & \frac{939457771}{4662187500} & -\frac{168763034}{388515625} & \frac{333046763}{1554062500} & \frac{19629003023}{18648750000} \end{pmatrix},$$

$$Q = \begin{pmatrix} 1 & 0 & 0 & 0 \\ \frac{1015625}{120733} & 1 & 0 & 0 \\ \frac{7376452890625}{53619698494} & -\frac{405}{14} & 1 & 0 \\ -\frac{4755875950106507681468751}{51052091899348840572958} & \frac{241922892409}{78349451754} & -\frac{32713015625}{350542022097} & 1 \end{pmatrix}. \tag{6.27}$$

References

[1] C. Bendtsen. A parallel stiff ODE solver based on MIRKs. *Adv. in Comp. Math.*, 7:27–36, 1997.

[2] J.R. Cash. On the integration of stiff ODEs using extended backward differentiation formulae. *Numer. Math.*, 34:235–246, 1980.

[3] J.R. Cash. The integration of stiff initial value problems in ODEs using modified extended backward differentiation formulae. *Comput. Math. Appl.*, 5:645–657, 1983.

[4] J.R. Cash and S. Considine. An MEBDF code for stiff initial value problems. *ACM Transactions on Mathematical Software*, 18:142–155, 1992. Code available at `http://www.ma.ic.ac.uk/~jcash/IVP_software/finaldae/readme.html`.

[5] J. de Swart. *Parallel Software for Implicit Differential Equations.* PhD thesis, Universirt of Amsterdam, 1997.

[6] G. Denk and P. Rentrop. Mathematical models in electric circuit simulation and their numerical treatment. In *Proc. of the Conference 'Numerical Treatment of Differential Equations', Halle (DDR).* Teubner, 1989.

[7] J.E. Frank and P.J. van der Houwen. Parallel iteration of backward differentiation formulas. Submitted for publication, 1999.

[8] E. Hairer, C. Lubich, and M. Roche. *The numerical solution of differential-algebraic systems by Runge-Kutta methods.* Lecture Notes in Mathematics, 1409. Springer-Verlag, 1989.

[9] E. Hairer and G. Wanner. *Solving ordinary differential equations, II. Stiff and differential-algebraic problems.* Springer-Verlag, Berlin, 1991.

[10] E. Hairer and G. Wanner. Radau. Available at `ftp://ftp.unige.ch/pub/doc/math/stiff/radau.f`, 1998.

[11] E.H. Horneber. *Analysis of of nonlinear RCLÜ circuits by means of mixed potential functions and a systematic representation of nonlinear dynamic circuit analysis (German).* PhD thesis, University of Kaiserslautern, 1976.

[12] W. Hundsdorfer. On the error of general linear methods for stiff dissipative differential equations. *IMA J. Numer. Anal.*, 14:363–379, 1994.

[13] P. Kaps. Rosenbrock-type methods. In G. Dahlquist and R. Jeltsch, editors, *Numerical methods for stiff initial value problems*, Bericht nr. 9. Inst. für Geometrie und Praktische Mathematik der RWTH Aachen, 1981.

[14] W.M. Lioen and J.J.B. de Swart. Test set for IVP solvers, Release 2.0. Available at `http://www.cwi.nl/cwi/projects/IVPtestset/`, 1998.

[15] G.-Y. Psihoyios and J.R. Cash. A stability result for general linear methods with characteristic function having real poles only. *BIT*, 38:612–617, 1998.

Appendix A

The parallel/multiblock structure of DeFT

Abstract. In this appendix we review the relevant concepts from domain decomposition theory, and derive the methods used in DeFT. We also describe the rationale behind the parallel model and multiblock structure as implemented in the software.

A.1 Domain decomposition methods

Domain decomposition methods are iterative methods for solving large, domain-based problems by breaking them down into a number of simpler subproblems. As a software design choice, the use of domain decomposition methods can have several motivations:

- *Complex geometries* can be decomposed into simpler ones. This was historically the motivation for domain decomposition methods.
- *Interface problems.* For problems involving two or more media sharing common interfaces, domain decomposition methods provide a framework for combining special solvers for each medium. In fluid dynamics, for example, one could use an Euler solver in inviscid regions and a Navier-Stokes code in the boundary layer.
- *Local grid refinement* can be accomplished using patches.
- *Memory limitations.* For very large discretizations, domain decomposition together with distributed computing may provide a solution to single processor memory limitations.
- *Parallel computing.* Domain decomposition methods can be used as parallel preconditioners for solving more general linear systems on parallel computers.

The general coordinate discretization used in DeFT assumes the domain to be topologically similar to a rectangle. To be able to handle more complex regions, domain decomposition was introduced.

Domain decomposition methods often have a great deal of inherent coarse-grained parallelism. The speedup of porting a sequential domain decomposition method to a parallel computer may be quite high: we have observed parallel efficiencies of 80% or better.

There are two standard approaches to domain decomposition methods, namely *Schur complement methods* and *Schwarz methods*, and much work has been done to unify these. As a result, there are number of cases in which the two approaches are shown to be equivalent. Schur complement methods, or *iterative substructuring methods*, are algebraic methods based on eliminating the unknowns in the subdomain interiors, reducing the problem to a computation of interface unknowns. Schwarz methods are grid-motivated methods based on iteratively correcting the error on the various subdomains.

A.1.1 Notation and conventions

At the heart of any software for solving partial differential equations is a linear system solver (or more likely, a whole arsenal of solvers) for systems of the form

$$Au = f, \tag{A.1}$$

where A is a $d \times d$ matrix and u and f are d-vectors. In DeFT, the matrix A represents the discrete momentum equations, pressure correction or transport equations and is very sparse; f represents any forcing terms and boundary conditions; and u is the solution to the given equation. We will take (A.1) as the starting point for our discussion.

A.1.1.1 Reorderings

Let P be a row- or column-wise permutation of the identity matrix. Define the *reordering* of the system (A.1) according to P by

$$\hat{A}\hat{u} = \hat{f}, \quad \hat{A} = P^T A P,\ \hat{u} = P^T u,\ \hat{f} = P^T f.$$

In general we will freely use reorderings without explicitly mentioning the permutation matrix. (We will assume the equations and unknowns were written down in the right order to begin with!)

A.1.1.2 Decompositions

Assume that the linear system (A.1) results from discretization of a partial differential equation defined on a domain Ω using a finite volume method. Let Ω be decomposed into a set of m subdomains $\Omega_i \subset \Omega$, $i = 1, \ldots, m$ such that $\Omega \subset \bigcup_{i=1}^{m} \Omega_i$. Initially, we will also assume that the subdomains do not overlap, i.e. that $\Omega_i \cap \Omega_j = \emptyset$, for $i \neq j$. A two-subdomain example is shown in Figure A.1.

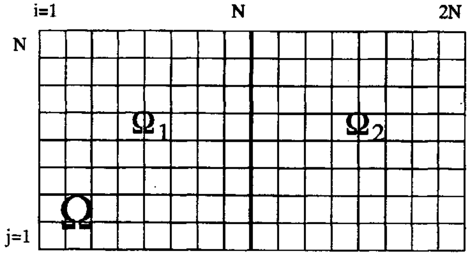

FIGURE A.1: A domain Ω composed of two non-overlapping subdomains.

A.1.1.3 Duplicated boundary unknowns

With a non-overlapping domain decomposition, if a cell-centered discretization is used, it is possible to reorder the system (A.1), grouping together all unknowns that share a common subdomain, to obtain the block system

$$\begin{bmatrix} A_{11} & \cdots & A_{1m} \\ \vdots & \vdots & \vdots \\ A_{m1} & \cdots & A_{mm} \end{bmatrix} \begin{pmatrix} u_1 \\ \vdots \\ u_m \end{pmatrix} = \begin{pmatrix} f_1 \\ \vdots \\ f_m \end{pmatrix}. \tag{A.2}$$

For vertex- or edge-based discretizations (or staggered-), there is an ambiguity with unknowns located on the inter-subdomain boundary itself. In this case we will duplicate the unknowns on the boundary, including a copy within each set of subdomain unknowns to obtain an augmented system. For example, consider a decomposition into two subdomains, using an edge-based discretization. Denote the unknowns strictly in the first and second subdomains by subscripts 1 and 2, and denote the unknowns on the interface boundary with a subscript $\mathcal{B}$. The block system becomes

$$\begin{bmatrix} A_{11} & A_{1\mathcal{B}} & A_{12} \\ A_{\mathcal{B}1} & A_{\mathcal{B}\mathcal{B}} & A_{\mathcal{B}2} \\ A_{21} & A_{2\mathcal{B}} & A_{22} \end{bmatrix} \begin{pmatrix} u_1 \\ u_{\mathcal{B}} \\ u_2 \end{pmatrix} = \begin{pmatrix} f_1 \\ f_{\mathcal{B}} \\ f_2 \end{pmatrix}.$$

Augmenting this system with duplicate interface unknowns gives

$$\begin{bmatrix} A_{11} & A_{1\mathcal{B}} & 0 & A_{12} \\ A_{\mathcal{B}1} & A_{\mathcal{B}\mathcal{B}} & 0 & A_{\mathcal{B}2} \\ A_{\mathcal{B}1} & 0 & A_{\mathcal{B}\mathcal{B}} & A_{\mathcal{B}2} \\ A_{21} & 0 & A_{2\mathcal{B}} & A_{22} \end{bmatrix} \begin{pmatrix} u_1 \\ u_{\mathcal{B}} \\ u'_{\mathcal{B}} \\ u_2 \end{pmatrix} = \begin{pmatrix} f_1 \\ f_{\mathcal{B}} \\ f_{\mathcal{B}} \\ f_2 \end{pmatrix},$$

where $u_{\mathcal{B}} \equiv u'_{\mathcal{B}}$ if $A_{\mathcal{B}\mathcal{B}}$ is nonsingular. This system can now be written as a 2×2 block system (A.2) with blocks corresponding to the partitioning $\tilde{u}_1 = \binom{u_1}{u_{\mathcal{B}}}$ and $\tilde{u}_2 = \binom{u'_{\mathcal{B}}}{u_2}$. We will assume that this is done whenever necessary.

A.1.1.4 Interface unknowns

Let a_{ij} denote the element of A in row i and column j. If $a_{ij} \neq 0$, we say ***unknown i is coupled to unknown j***. Similarly if block $A_{ij} \neq 0$ in (A.2), we say ***subdomain i is coupled to subdomain j***. Define the ***interface unknowns*** as the set of unknowns in a given subdomain which are coupled to unknowns in another subdomain.

A.1.2 Schwarz methods

A.1.2.1 Non-overlapping Schwarz iterations

The original Schwarz methods involved alternately solving a series of boundary value problems on overlapping domains, with the process repeating until convergence.

For matrices the boundary value problems are replaced by block rows. Consider first the non-overlapping case of the domain in Figure A.1, with block system

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \end{pmatrix}.$$

Given an initial guess $u^{(0)}$, we produce a series of iterates $u^{(k)}$, $k = 1, 2, \ldots$, by solving for $u_1^{(k)}$ from

$$A_{11}u_1^{(k)} = f_1 - A_{12}u_2^{(k-1)}, \tag{A.3}$$

(i.e. using the value of $u_2^{(k-1)}$ as a boundary condition) and then similarly solving for $u_2^{(k)}$ from

$$A_{22}u_2^{(k)} = f_2 - A_{21}u_1^{(k)}. \tag{A.4}$$

This process is referred to as the ***multiplicative Schwarz iteration***. Alternatively, we can replace $u_1^{(k)}$ in (A.4) with $u_1^{(k-1)}$, allowing us to solve for $u_1^{(k)}$ and $u_2^{(k)}$ concurrently. The resulting iteration is referred to as the ***additive Schwarz iteration***.

A.1.2.2 Overlapping Schwarz iterations

More generally, one can consider subdomains with some overlap. Roughly speaking, the iteration (A.3)-(A.4) may be modified such that the operators A_{11} and A_{22} represent the discretizations in the interiors of the overlapping subdomains, and right hand side operators A_{12} and A_{21} represent interpolation operators coupling the solution to the adjacent subdomain.

Many variations are possible. For example:

- Grids on different subdomains may be non-matching in the overlap region
- The discrete operators A_{11} and A_{22} may result from very different discretization methods.
- Artificial boundary conditions may be mixed and nonsymmetric.

The convergence rate may be made independent of grid refinement by using a fixed overlap in physical units (for example, if the grid resolution is halved, then the number of cells of overlap must be doubled).

A.1.2.3 Multilevel methods

To achieve a convergence rate that is independent of the number of subdomains, it is necessary to have global communication of the error. This is accomplished by projecting the residual onto a global space, typically with a much coarser grid, solving for the error on that space and interpolating a correction back to the subdomains. This process can be done recursively, by using several intermediate levels with successively fewer subdomains, and is related to multigrid methods. There are obviously many challenges involved in producing an efficient implementation of subspace correction.

A.1.3 Schur complement methods

In this section we assume a non-overlapping decomposition. We apply a reordering to the system (A.1), grouping all interface unknowns together into $u_{\mathcal{B}}$ and all remaining interior unknowns into $u_{\mathcal{I}}$ to obtain

$$\begin{bmatrix} A_{\mathcal{II}} & A_{\mathcal{IB}} \\ A_{\mathcal{BI}} & A_{\mathcal{BB}} \end{bmatrix} \begin{pmatrix} u_{\mathcal{I}} \\ u_{\mathcal{B}} \end{pmatrix} = \begin{pmatrix} f_{\mathcal{I}} \\ f_{\mathcal{B}} \end{pmatrix},$$

where it is assumed that unknowns in $u_{\mathcal{I}}$ are grouped together according to subdomain so that $A_{\mathcal{II}}$ is block diagonal.

The matrix A can then be factored as

$$A = \begin{bmatrix} I & 0 \\ A_{\mathcal{BI}}A_{\mathcal{II}}^{-1} & I \end{bmatrix} \begin{bmatrix} A_{\mathcal{II}} & 0 \\ 0 & S \end{bmatrix} \begin{bmatrix} I & A_{\mathcal{II}}^{-1}A_{\mathcal{IB}} \\ 0 & I \end{bmatrix},$$

with $S = A_{\mathcal{BB}} - A_{\mathcal{BI}}A_{\mathcal{II}}^{-1}A_{\mathcal{IB}}$, the Schur complement of the interface unknowns. Solving for A with this factorization involves solving several systems with the block-diagonal matrix $A_{\mathcal{II}}$, which can be done cheaply in parallel. In addition a subsystem

$$Sv = g \tag{A.5}$$

must be solved. Schur complement domain decomposition methods are based on the idea of solving (A.5) in parallel using a Krylov subspace method, for which S is not needed explicitly, but only its action upon v. Each multiplication of S with a vector requires an additional solve of the block diagonal system with matrix $A_{\mathcal{II}}$.

It is easy to show in the symmetric case that the spectral condition number of S is no worse than that of A:

$$\lambda_{\min}(S) = \min_{x_{\mathcal{B}} \neq 0} \frac{x_{\mathcal{B}}^T S x_{\mathcal{B}}}{x_{\mathcal{B}}^T x_{\mathcal{B}}^T} = \min_{x_{\mathcal{B}} \neq 0; A_{\mathcal{II}} x_{\mathcal{I}} + A_{\mathcal{IB}} x_{\mathcal{B}} = 0} \frac{x^T A x}{x_{\mathcal{B}}^T x_{\mathcal{B}}} \geq \min_{x \neq 0} \frac{x^T A x}{x^T x} = \lambda_{\max}(A).$$

An analogous argument shows $\lambda_{\max}(S) \leq \lambda_{\max}(A)$, giving the desired result. In practice the condition number of the Schur complement is much better than this suggests: while the condition number of A increases as $\mathcal{O}(h^{-2})$ for two dimensional problems, that of S only increases as $\mathcal{O}(h^{-1})$. The convergence is still usually not satisfactory, however, and something must be done to precondition the system (A.5).

A.1.4 Convergence theory

Although much can and has been said about the convergence of domain decomposition methods applied to finite element discretizations, there exists no satisfying general convergence theory for domain decomposition methods applied to finite volume or finite difference methods. Chang and Schultz[2] provide convergence results for block diagonal preconditioners. Tang [5] also discusses convergence theory for certain finite difference discretizations in one- and two-dimensions.

A.2 Domain decomposition methods used in DeFT

In this section we will describe the two domain decomposition methods implemented in DeFT, and show that to a certain degree, they are equivalent.

Remark. *Within the documentation and source code of DeFT, the Schur complement and approximate Schwarz domain decomposition procedures are incorrectly referred to as the "accurate" and "inaccurate" solution methods, respectively. In the first place, this is not in agreement with the standard terminology ("exact" and "approximate") for approximate subdomain solution methods; and in the second place, it is incorrect usage of "accuracy," when what is actually intended is "precision." Nevertheless, we will continue to use this nomenclature in this Appendix for the sake of agreement with the rest of the documentation.*

A.2.1 The Schwarz method with approximate solver (a.k.a. "Inaccurate solution")

Notice that the iteration (A.3)-(A.4) is a classical iteration of the form

$$Ku^{(k)} = f - (A - K)u^{(k-1)}, \tag{A.6}$$

where $K \equiv K_G$ for multiplicative Schwarz, $K \equiv K_J$ for additive Schwarz, and

$$K_G = \begin{bmatrix} A_{11} & 0 \\ A_{21} & A_{22} \end{bmatrix}, \quad K_J = \begin{bmatrix} A_{11} & 0 \\ 0 & A_{22} \end{bmatrix}$$

are the block Gauss-Seidel and block Jacobi preconditioners, respectively. When solving the system (A.6) on a parallel computer, it is clear that K_J offers more opportunity for parallelism than K_G. For this reason, the parallel method used in DeFT is additive Schwarz, while multiplicative Schwarz is used as the sequential method.

For the more general m-subdomain case of (A.2), the block Jacobi preconditioner becomes

$$K_J = \begin{bmatrix} A_{11} & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & A_{mm} \end{bmatrix}.$$

In a real implementation of Schwarz iteration, one would normally solve equation (A.2) not with the iteration (A.6), but instead using a Krylov subspace methods such as CG or GMRES, and using K_J or K_G as a preconditioner.

When the block Jacobi preconditioner is used in cooperation with a Krylov subspace method, a system of the form $K_J v = r$ must be solved in each iteration, i.e. m systems of the form $A_{ii} v_i = r_i$ must be solved (in parallel on m processors, preferably). In as much as the code already relies on iterative methods, and since in many cases the blocks A_{ii} have a similar sparsity pattern to A, it makes sense to solve the blocks also using an iterative method. One is immediately confronted with the question of how accurately to solve the subproblems. Our experience with the block Jacobi preconditioner in DeFT suggests that the answer is: *very approximately.* We have considered solving the subproblems to fixed tolerances of 10^{-1}, 10^{-2} and 10^{-6}, as well as simply applying a single iteration with a local incomplete factorization preconditioner, with the result that in most cases the incomplete factorization preconditioner is a good enough approximation, and is relatively so cheap, that it outperforms any iteration. In the few cases where iteration is faster, a tolerance of 10^{-1} is the winner.

A.2.2 The Schur complement method (a.k.a. "Accurate solution")

Consider again the two subdomain case of Figure A.1, and the iteration (A.6). Let the unknowns be re-ordered within each subdomain such that the interface unknowns come last. Indicating equations corresponding to subdomain i by a parenthesized superscript i, the system (A.2) can be written as

$$A = \begin{bmatrix} A^{(1)}_{II} & A^{(1)}_{IB} & 0 & 0 \\ A^{(1)}_{BI} & A^{(1)}_{BB} & 0 & A^{(1)}_{12} \\ 0 & 0 & A^{(2)}_{II} & A^{(2)}_{IB} \\ 0 & A^{(2)}_{21} & A^{(2)}_{BI} & A^{(2)}_{BB} \end{bmatrix} \begin{pmatrix} u^{(1)}_I \\ u^{(1)}_B \\ u^{(2)}_I \\ u^{(2)}_B \end{pmatrix} = \begin{pmatrix} f^{(1)}_I \\ f^{(1)}_B \\ f^{(2)}_I \\ f^{(2)}_B \end{pmatrix}, \tag{A.7}$$

where the non-zero structure of A is due to the definition of interface unknowns. The additive Schwarz preconditioner for this system is

$$K_J = \begin{bmatrix} A_{II}^{(1)} & A_{IB}^{(1)} & 0 & 0 \\ A_{BI}^{(1)} & A_{BB}^{(1)} & 0 & 0 \\ 0 & 0 & A_{II}^{(2)} & A_{IB}^{(2)} \\ 0 & 0 & A_{BI}^{(2)} & A_{BB}^{(2)} \end{bmatrix}$$

Rewriting (A.6) in the form $u^{(k)} = -K_J^{-1}(A - K_J)u^{(k-1)} + K_J^{-1}f$, we find the additive Schwarz iteration is equivalent to

$$\begin{pmatrix} u_I^{(1)} \\ u_B^{(1)} \\ u_I^{(2)} \\ u_B^{(2)} \end{pmatrix} = \begin{pmatrix} A_{II}^{(1)^{-1}} A_{IB}^{(1)} S^{(1)^{-1}} A_{12}^{(1)} u_B^{(2)} \\ -S^{(1)^{-1}} A_{12}^{(1)} u_B^{(2)} \\ A_{II}^{(2)^{-1}} A_{IB}^{(2)} S^{(2)^{-1}} A_{12}^{(2)} u_B^{(1)} \\ -S^{(2)^{-1}} A_{12}^{(2)} u_B^{(1)} \end{pmatrix} + K_J^{-1} f, \tag{A.8}$$

where $S^{(i)} = A_{BB}^{(i)} - A_{BI}^{(i)} A_{II}^{(i)^{-1}} A_{IB}^{(i)}$, from which it is clear that the iteration is independent of the unknowns in the subdomain interiors $u_I^{(i)}$. Thus the additive Schwarz iteration is effectively an iteration on the interface unknowns. This is what is done in the Schur complement method (the so-called "accurate solution" method) of DeFT. Concentrating only on the effective iteration on interface unknowns, the iteration (A.6) becomes

$$\begin{pmatrix} u_B^{(1)} \\ u_B^{(2)} \end{pmatrix}^{(k)} = \begin{pmatrix} -S^{(1)^{-1}} A_{12}^{(1)} u_B^{(2)} \\ -S^{(2)^{-1}} A_{12}^{(2)} u_B^{(1)} \end{pmatrix}^{(k-1)} + \begin{pmatrix} S^{(1)^{-1}} f_B^{(1)} - A_{BI}^{(1)} A_{II}^{(1)^{-1}} f_I^{(1)} \\ S^{(2)^{-1}} f_B^{(2)} - A_{BI}^{(2)} A_{II}^{(2)^{-1}} f_I^{(2)} \end{pmatrix}. \tag{A.9}$$

To see that in fact this is a Schur complement method as described in Section A.1.3, reorder (A.7) above, placing all interior nodes first and all interface nodes last, we obtain the system

$$A = \begin{bmatrix} A_{II}^{(1)} & 0 & A_{IB}^{(1)} & 0 \\ 0 & A_{II}^{(2)} & 0 & A_{IB}^{(2)} \\ A_{BI}^{(1)} & 0 & A_{BB}^{(1)} & A_{12}^{(1)} \\ 0 & A_{BI}^{(2)} & A_{21}^{(2)} & A_{BB}^{(2)} \end{bmatrix} \begin{pmatrix} u_I^{(1)} \\ u_I^{(2)} \\ u_B^{(1)} \\ u_B^{(2)} \end{pmatrix} = \begin{pmatrix} f_I^{(1)} \\ f_I^{(2)} \\ f_B^{(1)} \\ f_B^{(2)} \end{pmatrix}.$$

By eliminating the interior unknowns, we arrive at the Schur complement system for the interface unknowns

$$S u_B = \hat{f}_B,$$

with

$$S = \begin{bmatrix} S^{(1)} & A_{12}^{(1)} \\ A_{21}^{(2)} & S^{(2)} \end{bmatrix}, \quad u_{\mathcal{B}} = \begin{pmatrix} u_{\mathcal{B}}^{(1)} \\ u_{\mathcal{B}}^{(2)} \end{pmatrix}, \quad \hat{f}_{\mathcal{B}} = \begin{pmatrix} f_B^{(1)} - A_{\mathcal{II}}^{(1)^{-1}} A_{\mathcal{IB}}^{(1)} f_{\mathcal{I}}^{(1)} \\ f_B^{(2)} - A_{\mathcal{II}}^{(2)^{-1}} A_{\mathcal{IB}}^{(2)} f_{\mathcal{I}}^{(2)} \end{pmatrix}.$$

Defining

$$K_S = \begin{bmatrix} S^{(1)} & 0 \\ 0 & S^{(2)} \end{bmatrix},$$

it is clear that (A.9) is equivalent to the iteration

$$u_{\mathcal{B}}^{(k)} = -K_S^{-1}(S - K_S)u_{\mathcal{B}}^{(k-1)} + K_S^{-1}f. \tag{A.10}$$

The approach of this section can be related to the discussion of Section A.1.3, in which it was said that a preconditioner was still needed for good convergence of the Schur complement system (A.5). With Krylov subspace acceleration of the "accurate" solution method, the preconditioner used is K_S. The Schur complement system is explicitly preconditioned in DeFTthrough the use of (A.10).

The equivalence of the two approaches used in DeFT is intentional, see [1]. A consequence of this equivalence is that the convergence of the "accurate" and "inaccurate" methods in DeFT should be the same when the "inaccurate" method is solved to high tolerance.

The extension of the discussion of this section to decompositions with more subdomains or to the multiplicative Schwarz iteration is straightforward.

The equivalence between the Schwarz and Schur approaches breaks down with approximate subdomain solution. This can be attributed to the fact that with approximate solution of the subdomains, (A.8) will no longer be independent of the unknowns in the interior of the subdomains.

A.3 Parallel and multiblock substructures

Execution of DeFT on a parallel machine implies using domain decomposition, i.e. the multiblock functionality of the software. The user is responsible for defining a decomposition of the domain into blocks, and the blocks will be distributed over available processors.

As discussed in the previous section, there are two multiblock methods in DeFT: accurate (i.e. Schur complement) and inaccurate (i.e. Schwarz) methods.

The accurate multiblock method solves the reduced equations for the interface variables on a designated *host* process, where the matrix-vector multiplication with the explicitly-preconditioned Shur complement matrix requires a Schwarz iteration on the subdomains. The subdomain solutions are performed on *nodes*, referring to the remaining processes. The node tasks isndd in this case are "dumb" subdomain solvers. They simply enter a loop—receiving boundary conditions from the host, solving the subdomain problem, and returning the interface variables to the host—until told to stop. All communications are between host and node.

The accurate multiblock method may be unaccelerated or accelerated. Unaccelerated means the iteration (A.9) is repeated until convergence, accelerated means this iteration is performed once per global iteration of GMRESR (isblkgcr), as an explicitly-preconditioned matrix-vector multiplication (isaccma).

The inaccurate multiblock method performs the iteration (A.6) once per global iteration of GMRESR (isblkgcr), as a *preconditioner* (isblkpre). In this case, rather than a host-node model, all processes are autonomous, virtual data is exchanged with neighbors, and all processes participate in global operations such as inner products.

The function isdd determines what functions should be called for the desired type of domain decomposition method. The available algorithms are:

- unaccelerated accurate method (host calls ishplain)
- accurate multiblock method (host calls isblksol)
- inaccurate multiblock method (all processes call isblksol)

Remark. *Given the equivalence of the accurate and inaccurate methods shown in the preceding section, it is desirable to remove the less efficient accurate method from the code. However, the unaccelerated accurate method is used by the for multiphase flow part of the code.*

Figure A.2 shows a function call tree of for the multiblock structure. There are actually many (very many) more functions involved than are indicated in the figure, but we list only the main ones. The additional functions are called as subroutines by those discussed here.

A.3.1 Single block solution and virtual cell exchange

A single block domain in DeFT must be topologically rectangular. An extra row of "virtual" cells outside of the single block grid are used to implement boundary conditions. In this way, the stencil can be fixed for all unknowns, and values placed in the virtual cells to effect the desired boundary condition. For multiblock (domain decomposition) problems, the subdomains use this same structure.

In this case, however, the virtual cells along boundaries interfacing two subdomains are used to implement the domain decomposition. By copying data from neighboring subdomains into the corresponding virtual cells, and then solving the local subdomain

```
issbstep
├─ ...
├─ (issdd)
└─ isdd
    if (exact subdomain solution)
       if (host process)
          if (unaccelerated)
             ishplain      - additive Schwarz iteration
                (isreldd)
                (ishdd)
                (isddrel)
          if (accelerated)
             (isblksol)
       if (node process)
          isndd            - node side accurate domain decomposition
             isrcddnd      - receive virtual cell data on node
             (issdd)
             issnddnd      - send interface data to host
    if (inaccurate subdomain solution)
       (isblksol)
```

```
issdd           - local Schwarz iteration
├─ islinsol     - linear system solve with virtual cell data
│  ├─isuprhs    - eliminate virtual cell data
│  └─islsol     - linear solve without virtual cell data
└─ ismblk       - exchange local virtual cell data
```

```
ishdd         - host side accurate domain decomposition
├─ issnddhs - send virtual cell data to nodes
├─ (issdd)
└─ isrcddhs - receive interface data from nodes
```

```
isblksol                - Krylov subspace acceleration (GCR)
├─ (isreldd/isddrel)
├─ (issoldd/isddsol)
└─ isblkgcr             - actual GCR algorithm
   ├─ isblkmat          - matrix-vector product wrapper
   │  ├─isaccma         - accurate matrix-vector product, explicit preconditioning
   │  │  ├─ isreldd     - inject interface data into virtual cells
   │  │  ├─ (ishdd)
   │  │  └─ isddrel     - restrict virtual cells into interface array
   │  └─isgloma         - inaccurate matrix-vector product
   │     └─ (isddsol)
   └─ isblkpre          - preconditioner wrapper
      ├─isddsol         - inject contiguous vector into block interiors
      │  └─ ismbexch    - exchange virtual cell data
      ├─(issdd)
      └─issoldd         - restrict block interiors into contiguous vector
```

FIGURE A.2: Function call tree for multiblock structure

problem using the single block algorithm, one effectively performs an operation of the form (A.3).

Consider a linear system of the form

$$\begin{bmatrix} A_{11} & A_{12} \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = f_1,$$

where x_1 represents the set of unknowns in the local subdomain, x_2 represents *known* values in the virtual cells, and f_1 represents the right hand side for the local subdomain, already corrected for physical boundary conditions, if applicable. Given this situation, the function islinsol performs the following operations:

1. Update the right hand side with a linear combination of the virtual cells and extract the relevant block (A_{11}) from the matrix.
2. Solve the reduced system: $A_{11}x_1 = \hat{f}_1$.
3. Inject the solution x_1 back into the original data structure, restoring the original matrix and right hand side.

This is similar to the operation (A.3) of Section A.1.2.1.

The functions ismblk and ismbexch effect the exchange of data between neighboring subdomains: ismblk may be used to copy the interior unknowns from an individual block to the corresponding virtual cells of a neighboring block on the same processor; ismbexch does this for all blocks on all processors. Additionally, ismbexch may be called to compute the average of the ambiguous normal fluxes on the subdomain boundaries after convergence. This has been found necessary for conservation of mass.

A.4 Parallel substructure of DeFT

The parallel substructure of DeFT is based on a Single Program Multiple Data (SPMD) model and domain decomposition (see Section A.3 as well). In the SPMD model, all processes take responsibility for one or more subdomain, and execute the same program. There are a few tasks requiring deviation from the model—for example, output and error handling—which are handled by a single process designated as the *host*.

A.4.1 MPI

Communication between processes is achieved using function calls to the MPI library. A subset of the library is available in DeFT through calls to wrapper functions. In this way, the parallel option of the code can be disabled with minimal modification of the source for installations where either MPI is unavailable or where parallel operation is not desired.

MPI is the standard library for message passing parallel communication models. Implementations conforming to the standard are available on a variety of architectures, including workstations, and several good implementations are available for free. The use of MPI increases portability.

Although MPI allows the definition of *communication contexts*, so that processes sharing a communicator can be shielded from those in another communicator, we currently make use of only the global communicator (MPI_COMM_WORLD) in DeFT.

The following table lists the most important MPI functions used in DeFT, the name of the DeFT wrapper, and a short description of the function. See [4] or the MPI standard [3] for full definitions of these routines.

MPI function	DeFT function	Description
MPI_BCAST	issync	Broadcasts a message from one process to all other processes
MPI_ALLREDUCE	isglsum	Combines values from all processes, returning the result to all processes
MPI_ISEND	issenmpi	Initiates a send, returning control to the calling procedure
MPI_RECV	isrecmpi	Waits for and receives a message

MPI also defines a number of parameters. The following table lists the most important parameters of relevance to DeFT.

MPI Parameter	Description
MPI_COMM_WORLD	The top level communicator
MPI_SUCCESS	The return value of the error parameter if no error has occurred
MPI_ANY_SOURCE	Wildcard for receiving a message regardless of the source
MPI_ANY_TAG	Wildcard for receiving a message regardless of the message tag

A.4.2 Parallel common block cisparal

Some important parameters and global variables needed for the parallel implementation are defined in the common block include file cisparal.

There are three global variables used to define the define the communication network topology:

- iacnodes - the number of active nodes (processes)
- inodenr - the rank of the current node

- ISHOST - the rank of the node designated as host

Additionally, we define a number of unique message contexts. These are used as message tags in the corresponding communications to eliminate the possibility of a message being misinterpreted. The message contexts are:

Context name	value	description
ICERROR	99	Error message context
ICWARN	98	Warning message context
ICPRINT	97	Output context
ICEXNODE	96	Node has exited context
ICFIELD	6	Field communication context
ICVCDATA	5	Virtual cell data exchange context
ICVCINFO	4	Virtual cell info array context
ICSTCR	3	Steady state criterion check context
ICCORR	2	Pressure correction context
ICDEFLT	1	Default context

Finally, we define aliases for the default MPI datatypes, so that these may be changed, if necessary (due to architecture differences, for example). The types used in DeFT are:

DeFT type	MPI type
IS_ANY	MPI_ANY
ISCHAR	MPI_CHARACTER
ISDOUBLE	MPI_DOUBLE_PRECISION
ISINT	MPI_INTEGER

These datatypes are defined in the function istpar.

A.4.3 Basic send and receive model

The basic send/receive model utilizes non-blocking sends and blocking receives. This is done to allow some overlapping of computation and communication, where possible, and to prevent tie-ups in a situation where many processes are exchanging data (such as in the exchange of boundary information in the domain decomposition method).

A call to issenmpi returns immediately with an MPI request number. It is the programmer's responsibility to check that the message has actually been sent—through a call to iswaitall with the request number—before modifying the data in the send buffer (array).

A call to isrecmpi will not return until the message has actually been received.

If the sends were also blocking, then it would be necessary to coordinate sends and receives, making sure the order of communications was the same on all processes, to prevent a tie up.

A.4.4 Global communications

MPI defines some common functions involving global communications with a group of processes. Two of these used in DeFT are the broadcast and global reduction operations.

The function issync is a wrapper for MPI_BCAST. This function synchronizes the value of a scalar or array on all processes. For example, if the data for a computation is only available on one process, then the result can be distributed to all processes through a call to this function.

The function isglsum is a wrapper for MPI_ALLREDUCE with reduction operation MPI_SUM. This function performs a vector sum:

$$y = \sum_{i=1}^{p} x_i, \quad y, x_i \in \mathbb{R}^d$$

where x_i resides on process i. The result y is distributed to all p processes. This function is primarily used to sum the local contributions to an inner product, in which case $d = 1$.

A.4.5 Error handling and printing

For output to be deterministic, it should be performed by only one process, the designated host. If a node generates conditional output, a mechanism must be implemented on the host to receive and process such output. A similar problem occurs when there is an unrecoverable error on a process: the group must be alerted and stopped. Sending information to the designated host for output and stopping program execution when an error occurs are complicated in parallel by the fact that that the receiving process is not expecting the message.

To handle these situations, the receive function isrecmpi checks first for special messages before handling the requested receive. In this way, processes may be notified of unexpected actions at the first subsequent call to isrecmpi.

The logic of the error handling system is as follows:

1. If an error originates on a node, it sends a message with context tag ICERROR to the host.
2. If the host receives notification of an error from a node, or if an error occurs on the host, a stop signal is sent from the host to all nodes

3. If a node receives a stop signal, it terminates.

References

[1] E. Brakkee and P. Wilders. Schwarz and Schur: an algebraical note on equivalence properties. *SIAM J. Sci. Comput.*, 20(6):2297–2303, 1999.

[2] M. Y.-M. Chang and M. H. Schultz. Bounds on block diagonal preconditioning. *Parallel Algorithms and Applications*, 1:141–164, 1993.

[3] Massage Passing Interface Forum. MPI: A message-passing interface standard. *International Journal of Supercomputer Applications*, 8(3/4), 1994.

[4] M. Snir, S.W. Otto, S. Huss-Lederman, D.W. Walker, and J. Dongarra. *MPI: The Complete Reference.* MIT Press, 1996.

[5] W.P. Tang. Generalized Schwarz splittings. *SIAM J. Sci. Statist. Comput.*, 13(2):573–595, 1992.

Summary of *Efficient Algorithms for the Numerical Solution of Differential Equations*

The efficiency of a numerical integration method is defined to be the production, measured in digits of accuracy or qualitative agreement of the numerical result, divided by the cost, measured in computational effort or waiting time, of producing the result. The actual efficiency is often difficult to measure, but relative efficiencies can be compared in many cases. Three independent techniques yielding efficient methods are considered:

1. simplification of implicit relations by using implicit-explicit splittings,
2. exploitation of parallelism across the problem space,
3. exploitation of parallelism across the method parameters.

The advantage of implicit-explicit (IMEX) methods is that problems with both stiff and nonstiff terms can be integrated with only the stiff terms being treated implicitly. In some cases this leads to simplified implicit relations, for example, if the stiff terms are linear whereas the nonstiff terms are nonlinear—the case for the Navier-Stokes equations. All implicit-explicit linear multistep methods can be seen as the combination of an implicit multistep methods with an extrapolated explicit counterpart. In the simplest case of the IMEX Euler method, the stability condition is satisfied if the eigenvalues of the explicit terms are contained in the stability region of the related explicit method. This does not hold for general IMEX linear multistep methods, however. We consider two questions: 1) under what additional restriction on the eigenvalues of the explicit part does the method remain A-stable for the implicit part, and 2) under what restriction on the A(α)-stability of the implicit part do we maintain the stability region of the explicit part.

Parallelism across the problem space is especially accessible when solving partial differential equations, for which the coupling is often sparse and localized, allowing for easy load balancing and a small volume of communication. Since these problems are also frequently scalable, the resulting methods are suitable for implementation on massively parallel distributed memory computers. We consider parallel implementation of a simple block-diagonal preconditioner based on domain decomposition, in which the blocks are approximately solved using inner iterations. The global iteration method is GCR, and we discuss alternative orthogonalization methods to reduce the number of required global communications. Parallel timings are given for a Poisson equation, arising as a result of the pressure correction method, and the Bousinesq equations for natural convection in a heated cavity.

The degree of parallelism across the method parameters is proportional to the number of stages, and therefore is especially practical when high order accuracy is required. We consider a number of approaches to parallelizing the extended backward differentiation formulas, a class of general linear methods maintaining high accuracy in the presence of stiffness. We develop, analyze and test 3- and 4-processor implementations for up to 6th order L-stable methods. The two most promising approaches for parallelizing the (lower triangular) stage coefficient matrix are: 1) approximation by the diagonal and 2) staggering of the abscissa to obtain a diagonalizable matrix.

Jason Frank

Samenvatting van *Efficiente Algoritmen voor het Numeriek Oplossen van Differentiaalvergelijkingen*

De efficiëntie van een numerieke methode is gedefinieerd als de opbrengst, gemeten in het aantal cijfers nauwkeurigheid of in een kwalitatieve overeenkomst van de numerieke oplossing, gedeeld door de kosten, gemeten in rekenwerk of wachttijd, die nodig zijn om het resultaat te verkrijgen. De werkelijke efficiëntie is vaak moeilijk te meten, maar in veel gevallen kunnen relatieve efficiënties worden vergeleken. Drie onafhankelijke technieken, die efficiënte methoden opleveren, zullen aan de orde komen:

- vereenvoudiging van de impliciete relaties door het gebruik van impliciete-expliciete methoden,
- exploitatie van parallellisme in de probleemdimensie,
- exploitatie van parallellisme in de taken van de methode.

Het voordeel van impliciete-expliciete (IMEX) methoden is dat problemen met zowel stijve als niet-stijve termen geïntegreerd kunnen worden met een impliciete behandeling van alleen de stijve termen. Dit leidt tot vereenvoudiging van de impliciete relaties wanneer de stijve termen lineair zijn terwijl de niet-stijve termen niet-lineair zijn—zoals het geval is bij de Navier-Stokes vergelijkingen. Alle impliciete-expliciete lineaire meerstaps methoden kunnen worden gezien als de combinatie van een impliciete meerstaps methode met een geëxtrapoleerde expliciete tegenhanger. In het eenvoudigste geval van de IMEX Euler methode, wordt aan de stabiliteitsconditie voldaan wanneer de eigenwaarden van de expliciete termen binnen het stabiliteitsgebied van de verwante expliciete methode vallen. Dit geldt echter niet voor algemene IMEX lineaire meerstaps methoden. We hebben de volgende twee vragen nader beschouwd: 1) onder welke toegevoegde restricties aan de eigenwaarden van het expliciete deel blijft de methode A-stabiel voor het impliciete deel en 2) onder welke restricties voor de A(α)-stabiliteit van het impliciete deel behouden we het stabiliteitsgebied van het expliciete deel.

Parallellisme in de probleemdimensie is met name toegankelijk tijdens het oplossen van partiële differentiaalvergelijkingen, waarvoor de koppelingen vaak ijl en lokaal zijn. Dit zorgt voor een gemakkelijke *load-balancing* en een kleine hoeveelheid communicatie. Omdat deze problemen meestal ook schaalbaar zijn, zijn de resulterende methoden geschikt voor implementatie op massief geparalleliseerd gedistribueerde geheugen computers (*massively parallel distributed memory computers*). Wij beschouwen parallelle implementatie van een eenvoudige blok-diagonale preconditioner gebaseerd op domeindecompositie waarin de blokken onnauwkeurig opgelost worden met een sub-iteratie. De hoofd-iteratie gebruikt de GCR methode, en we bespreken alternatieve orthogonalisatie methoden die kunnen leiden tot een vermindering van het aantal globale communicaties. Parallelle meettijden zijn gegeven voor zowel een Poisson vergelijking (nodig voor het berekenen van de drukcorrectie methode) als de Boussinesq vergelijkingen voor natuurlijke convectie in een verwarmde ruimte.

De mate van parallellisme in de verschillende taken van de methode is evenredig met het aantal *stages*, en dus is vooral toepasbaar wanneer hoge orde van nauwkeurigheid vereist wordt. We bespreken een aantal benaderingen voor het parallelliseren van de extended backward differentiation formulas (EBDF), een klasse van algemene lineaire methoden die een hoge nauwkeurigheid behoudt in de aanwezigheid van stijfheid. We ontwikkelen, analyseren en testen drie- en vier-processor implementaties voor tot en met zesde orde L-stabiele methoden. Twee veelbelovende benaderingen voor het parallelliseren van de (laag-driehoekige) *stage*-coëfficiënt matrix zijn: 1) benadering door de diagonaal en 2) gebruik van ongelijk verdeelde abscissa waarden om een diagonaliseerbare matrix te verkrijgen.

Jason Frank

Curriculum vitae

The author was born in Hutchinson, Kansas, the United States of America on 21 February 1970. He attended Great Bend High School from 1985 through 1988, and the University of Kansas from 1988 through 1995, earning a Bachelor of Science in Aerospace Engineering in 1992 and a Master of Science in Aerospace Engineering in 1994 with honors. In 1995 he came to the Netherlands where he spent six months as visiting researcher at the National Institute for Mathematics and Computer Science (CWI) in Amsterdam. Thereafter he received a four-year appointment (1996–99) as a Ph.D. student (AIO) at Delft University of Technology, during which time the majority of the research for this dissertation was carried out. Also during this appointment he again spent 18 months as guest researcher at CWI.