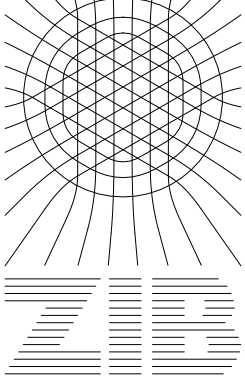
Konrad-Zuse-Zentrum für Informationstechnik Berlin
Takustr. 7, D-14195 Berlin - Dahlem



Rainald Ehrig

Ulrich Nowak

Peter Deuflhard

Highly Scalable Parallel Linearly-Implicit Extrapolation Algorithms

Technical Report TR 96-11 (December 1996)

Highly Scalable Parallel Linearly-Implicit Extrapolation Algorithms

Rainald Ehrig Ulrich Nowak Peter Deuflhard

Abstract

We present parallel formulations of the well established extrapolation algorithms EULSIM and LIMEX and its implementation on a distributed memory architecture.

The discretization of partial differential equations by the method of lines yields large banded systems, which can be efficiently solved in parallel only by iterative methods. Polynomial preconditioning with a Neumann series expansion combined with an overlapping domain decomposition appears as a very efficient, robust and highly scalable preconditioner for different iterative solvers. A further advantage of this preconditioner is that all computation can be restricted to the overlap region as long as the subdomain problems are solved exactly. With this approach the iterative algorithms operate on very short vectors, the length of the vectors depends only on the number of gridpoints in the overlap region and the number of processors, but not on the size of the linear system. As the most reliable and fast iterative methods based on this preconditioning scheme appeared GMRES or FOM and BICGSTAB. To further reduce the number of iterations in GMRES or FOM we can reuse the Krylov-spaces constructed in preceeding extrapolation steps.

The implementation of the method within the program LIMEX results in a highly parallel and scalable program for solving differential algebraic problems getting an almost linear speedup up to 64 processors even for medium size problems. Results are presented for a difficult application from chemical engineering simulating the formation of aerosols in industrial gas exhaust purification.

Contents

1 Introduction

In the last years we have seen an increasing demand for fast and high resolution simulations in many scientific and engineering areas. The analysis and optimization of large equipments in industrial chemical engineering as an example depends strongly on algorithms, which enable accurate solutions rapidly, otherwise the study of parameter dependencies would be impossible. Problems possessing multiple time scales provide the motivation for implicit algorithms and their inherent dynamical behaviour requires locally adaptive approaches. Furthermore very often a simulation process needs varying accuracies of the solutions, which makes iterative methods especially attractive. Nowadays the whole claim can in most cases only be satisfied by efficient parallel implementations.

Following this trends we study in this paper parallel methods for the solution of systems of partial differential equations, discretized by the method of lines, in the 1D case. Our basis is the well–established family of adaptive extrapolations codes EULSIM and LIMEX, see Deuflhard [16], Deuflhard and Nowak [17] and Deuflhard, Lang and Nowak [18], which cover a wide area of applications, e.g. combustion problems or catalytic processes. The common algorithmic principle of these programs is the linearly–implicit extrapolation method combined with an advanced order and stepsize control. Due to their linear structure, these methods do not need the solution of nonlinear systems and enable at every time step modifications in an adaptively defined grid. Therefore the underlying algorithms already fulfill the requirements for implicit discretization and local adaptivity, whereas this for accuracy dependent evaluation can only completely satified when using iterative linear solvers. As we will see, even the latter point will be accomplished in an efficient parallel realization.

Guidelines for our parallelization attempts were always the development of scalable and application independent algorithms, requiring only a minimum of constraints for a given problem, this is mainly a spatial locality property. On the other hand all our methods are targeted to tightly coupled massively parallel systems with distributed memory and high bandwidth communication, which clearly has determined some of our algorithmic decisions. As usual the right choice of a parallel preconditioner and an iterative solver predominantly decides, whether the battle for low computational cost and scalable parallelism can be won.

The outline of this paper is as follows. In §2 we briefly describe the linearly-implicit extrapolation algorithm. The general parallelization concepts and some algorithmic decisions are discussed in §3. §4 is devoted to the step by step evolution of the optimal parallel solver and preconditioner combination within the extrapolation codes. Several parallel implementation issues are explained in §5. Numerical results for a complex real application arising in chemical engineering are summarized in §6. Finally, we offer some general remarks and further expectations on the algorithms in §7.

2 Linearly-implicit extrapolation algorithms

For completeness, we summarize the derivation and assumptions of the linearly–implicit extrapolation algorithms. For a more thorough development, see [16, 17].

We consider a system of ordinary differential equations (ODEs)

$$y' = f(t,y)\ ,\quad y(t_0) = y_0\ ,\quad y \in \mathbb{R}^n \tag{1}$$

or a system of differential algebraic equations (DAEs) of the form

$$B(t,y)y' = f(t,y)\ ,\quad y(t_0) = y_0\ ,\quad y \in \mathbb{R}^n. \tag{2}$$

Herein the matrix B may be singular, but we assume that the system of DAEs is of index ≤ 1. Our main interest is concentrated on solving systems of ODEs and DAEs which are the result of a space discretization of a system of partial differential equations by the method of lines. Nevertheless our parallel methods should be applicable to other problem classes as well.

The linearly–implicit Euler discretization applied with stepsize h to the equations (1), (2) reads

$$y_{k+1} = y_k + (I - hA)^{-1} h f(y_k) \tag{3}$$

with $A = f_y(y)$ the Jacobian of f, respectively

$$y_{k+1} = y_k + (B(y_k) - hA)^{-1} h f(y_k) \tag{4}$$

with $A = (f(y) - By')_y$, the Jacobian of the residual of (2).

If an asymptotic h–expansion for the approximation error exists, the well known Richardson extrapolation can be used to construct approximations of higher order. Such expansions can be proven to exist for the linearly–implicit Euler method in general only for systems of ODEs. For systems of DAEs perturbed expansions still exist, but the extrapolation algorithm can further sucessfully be applied, even if the maximal attainable order may be reduced, see Lubich [33].

Let us define a basic stepsize H. Within the extrapolation method one computes approximations $T_{j,1}$ for $y(t_0 + H)$ using the described discretizations with stepsizes $h_j = H/n_j, j = 1, \ldots, j_{max}$. We have used the harmonic sequence $\{n_j\} = \{1, 2, 3, \ldots\}$, which has been reliable in all cases. Now the extrapolation table is given by the recursions

$$T_{j,k} = T_{j,k-1} + \frac{T_{j,k-1} - T_{j-1,k-1}}{n_j/n_{j-k+1} - 1}\ ,\quad k = 1, \ldots, j \tag{5}$$

and defines the higher order approximations $T_{j,k}$. As usual the so–called subdiagonal differences $\varepsilon_j = \|T_{j,j} - T_{j,j-1}\|$ are taken as error estimates using a weighted root mean square norm. If the error estimate is less than a prescribed tolerance, $T_{j,j}$ is accepted as an approximation for $y(t_0 + H)$ and the integration proceeds with a new estimated

optimal stepsize/order combination. An advanced adaptive stepsize and order control mechanism has been developed by DEUFLHARD [19] and is used within our algorithms without any significant changes.

The somewhat simplified algorithmic structure of the linearly–implicit extrapolation codes for one basic integration step can now be represented as follows.

ODEs (code EULSIM):

$$
\begin{aligned}
&\textit{Compute Jacobian } A = f_y(y_0) \\
&\textit{for } j = 1, \ldots, j_{max} \textit{ while convergence criterion not satisfied} \\
&\qquad h_j = H/n_j \\
&\qquad \textit{for } k = 0, \ldots, j-1 \\
&\qquad\qquad y_{k+1} = y_k + (I - h_j A)^{-1} h_j f(y_k) \qquad (6) \\
&\qquad T_{j,1} = y_j \\
&\qquad \textit{if } j > 1 \textit{ compute } T_{j,j} \textit{ and check convergence} \\
&y_{new} = T_{j,j}
\end{aligned}
$$

DAEs (code LIMEX):

$$
\begin{aligned}
&\textit{Compute Jacobian } A = (f(y_0) - By')_y \\
&\textit{for } j = 1, \ldots, j_{max} \textit{ while convergence criterion not satisfied} \\
&\qquad h_j = H/n_j \\
&\qquad \textit{for } k = 0, \ldots, j-1 \\
&\qquad\qquad y_{k+1} = y_k + (B(y_k) - h_j A)^{-1} h_j f(y_k) \qquad (7) \\
&\qquad T_{j,1} = y_j \\
&\qquad \textit{if } j > 1 \textit{ compute } T_{j,j} \textit{ and check convergence} \\
&y_{new} = T_{j,j}
\end{aligned}
$$

The schematic representations clearly exhibit that the efficiency of any serial or parallel implementation almost solely depends on the performance of two computational kernels: the evaluation of the Jacobian and the solution of the linear systems. It is also obvious that the main distinction between the two algorithms is the change of the matrix in the linear systems in every step of the innermost loop in the DAE code. Because in almost every real problem the matrix B covers only relatively few non–constant entries, instead of a standard LU factorization an iterative procedure may be preferable [17].

Define $\Delta B = B(y_0) - B(y_k)$ and the simple matrix splitting

$$B(y_k) - hA = (B(y_0) - hA) \; - \; \Delta B. \tag{8}$$

Using the corresponding relaxation scheme we get the following iterative algorithm.

$$\begin{aligned} \Delta_0 &= h(B(y_0) - hA)^{-1} f(y_k) \; , \\ y_{k+1}^0 &= y_k \; , \\ i = 1, 2, \ldots : & \\ \Delta_i &= (B(y_0) - hA)^{-1} \Delta B \; \Delta_{i-1} \; , \\ y_{k+1}^{i+1} &= y_{k+1}^i + \Delta_i \; . \end{aligned} \tag{9}$$

It is well known that such a relaxation scheme is equivalent to a fixed–point iteration on a preconditioned linear system. Here the preconditioner for the whole internal step is simply $B(y_0) - hA$. The algorithmic realization in LIMEX adaptively changes between this iterative method and the direct solution of the linear systems depending on a rough comparison of the respective amount of work. In any case at least the LU factorizations of the matrices $B(y_0) - h_j A$ are needed.

If we solve *all* linear systems by an iterative method (e.g. GMRES), a considerable simplification is possible. We can then exchange the relaxation scheme described above for this iterative solver, which is surely more effective, and use again a common preconditioner, but now instead of $B(y_0) - hA$ e.g. an ILU factorization for $B(y_0) - hA$ for the whole internal step. The resulting algorithmic structure looks very similar to the one of the ODE code. This technique makes iterative solvers very attractive even for serial implementations of LIMEX, but the topic is not further discussed in this paper. Whether this method is applicable also for parallel realizations, will be investigated in §4.4.

3 General parallelization concepts

As pointed out in the last section, the efficiency of any implementation of the extrapolation algorithms crucially depends on two computational tasks: the evaluation of the Jacobian and the solution of the linear systems. Therefore a parallel extrapolation code has to distribute this parts as far as possible.

At a first sight, a simple and promising method is to compute the approximations $T_{j,1}$ on different processors, because there evaluation is completely independent, but this incorporates some severe drawbacks. First the number of processors, which could be efficiently used, is restricted to the maximal order attainable by the extrapolation method. If we assign groups of different sizes of processors to the $T_{j,1}$, we have to construct different domain decompositions. Furthermore we could not use the original order and stepsize control techniques without changes, as in this procedure the used order j_{max} of (6) and (7) is determined within the current step. Moreover all processor groups need the whole Jacobian. Nevertheless this parallelization strategy, sometimes called "parallelization across method" , BURRAGE [11], is studied by some authors, especially by RAUBER [39], but seems meaningful only for explicit extrapolation methods, where the evaluation of the right–hand side of the ODEs is the main computational task.

A more general applicable and commonly used strategy is the "parallelization across the system" or domain decomposition method. This principle does not involve any limitation to the number of processors used in a parallel implementation.

The realization of a domain decomposition algorithm refers directly to properties of the underlying physical problem. It presupposes that the solution on the entire domain can be got from solutions on suitably selected subdomains. Within the context of the linearly–implicit extrapolation methods this principle requires that the function $f(t, y)$ in the right–hand side of the ODE (1) or DAE system (2) can be evaluated locally, i.e. that any value f_n only depends on y_n and on some y–values in the neighborhood. An equivalent to this locality property is that the Jacobian of f is a banded matrix and is therefore likewise computable in parallel. The assumption of local computability of f is naturally fulfilled for problems resulting from a discretization of PDE's by the method of lines. Now assume we could solve efficiently and in parallel the linear systems which arise during the extrapolation. Then it is obvious that the whole extrapolation scheme can be straightforward parallelized, if every processor manages a fixed segment of the approximations $T_{j,k}$. Apart from the exchange of boundary values, which is necessary for the evaluation of f on the processor boundaries, the extrapolation only requires global reduction routines for the dot products in the computation of the error estimates ε_k.

As in many other parallel numerical algorithms the efficiency of our parallel extrapolation implementation therefore strongly depends on the performance of the linear solver.

3.1 Direct versus iterative solution of the linear systems

At present the serial linearly–implicit extrapolation codes EULSIM and LIMEX use the LU factorization to solve the linear systems. Parallel implementations of the Gaussian algorithm have been studied by many authors, see e.g. in the textbooks by KUMAR ET AL. [30] or FREEMAN AND PHILLIPS [25]. There are highly scalable and efficient algorithms for large dense systems, but load balance and and perfomance are dependent on an appropriate mapping of the columns (or rows) of the matrix among the processors. They require a cyclic columnwise or better cyclic block–columnwise striped partitioning of the matrix, so that each processor is assigned to a number of contiguous blocks of columns. Such distributions do not cooperate with any meaningful domain decomposition and need the complete vector y_0 on each processor for the computation of the Jacobian. On a distributed memory machine this may be a significant drawback.

A second and more important disadvantage of the parallel Gaussian elimination is the loss of efficiency for banded systems, due to the imbalance of computation and communication. For banded linear systems there are some special direct approaches, an exhaustive survey and comparison has been made by ARBENZ AND GANDER [1]. Even if these methods use matrix partitionings referring to a domain decomposition, they show a reasonable behaviour only if the bandwidth of the matrices is *very* narrow. Furthermore none of the algorithms is scalable for high processor numbers. One of these direct methods will appear as a modification of the iterative algorithm we investigate and discuss in §4.4.

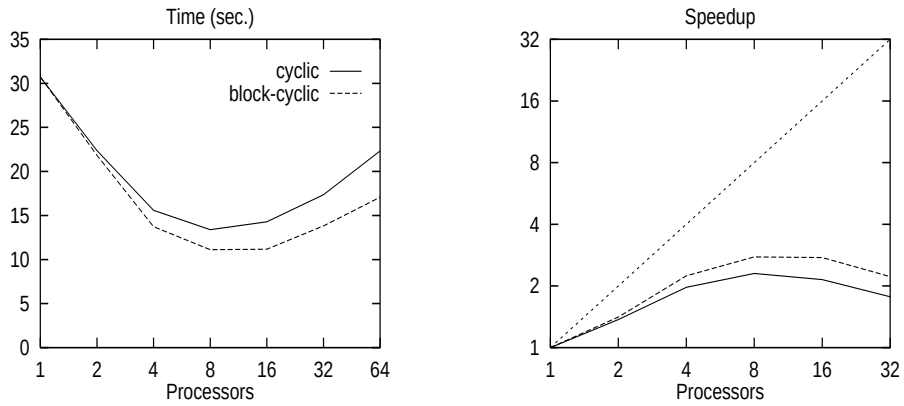


Figure 1: Performance results for a combustion test problem with direct parallel solvers for banded matrices

We have developed two versions of a parallel LU factorization using assembly coded Level 2 resp. Level 3 BLAS routines, based on cyclic columnwise and block–cyclic columnwise striped partitioning of the Jacobian (column storage with row pivoting scheme). In the latter we used a blocksize of 16 columns, which is an appropriate

setting on the Cray T3D. The communication was done via the Basic Linear Algebra Communication Subprograms (BLACS) for the Cray MPP system [21]. At the beginning of each integration step, every processor collects the whole vector f and computes indepedently his own columns of the Jacobian. After the parallel factorization the extrapolation is performed in parallel on a non–overlapping domain decomposition. The implementation of this technique in the program EULSIM exhibits clearly the limitation of the direct solution methods. Applied to a combustion problem, test problem 4 from PETERS AND WARNATZ [37], we get as maximal speedup only a factor of about 2.8 with 8 processors (block–cyclic distribution), see Figure 1. For higher processor numbers the computing times even increase. Nevertheless the parallelized Gaussian elimination can be a suitable method to solve very large systems of ODEs by the extrapolation method. This may be especially true if the Jacobian is not narrow banded and/or the systems are extremely stiff, which may lead to severe problems with iterative linear solvers.

As demonstrated the advantages of a parallel direct black–box solver are in most cases outweighted by its poor performance. Therefore we do not discuss direct solvers in more detail, but switch to iterative methods. The principal benefits of an iterative solution algorithm for the linearly–implicit extrapolation method can be summarized as follows:

- The partitioning of the Jacobian can refer to any domain decomposition, even for 2D or 3D problems.
- The accuracy of the solution of the linear systems can be adopted to the required accuracy of the solution of the integration.
- If the number of iterations does not "strongly" depend on the number of processors, the algorithms became scalable.
- Distributed evaluation and storage of the Jacobian is possible.

Although these properties may be convincing, all benefits using iterative methods strongly depend on an appropriate preconditioner. This key problem will be discussed in the next section.

4 Parallel iterative solvers

The general problem of finding an appropriate preconditioned iterative solver for a linear system $Ax = b$ is to look for a matrix P (the preconditioner) with the properties that $P^{-1}A$ is "good" conditioned and the system

$$P^{-1}Ax = P^{-1}b \tag{10}$$

is much easier to solve than the original system. Especially for parallel implementations there are no efficient black–box preconditioning techniques which can be applied to general matrices. Instead one has to construct problem dependent preconditioners which exploit special features of the considered problem class. The efficiency of an iterative solver on massively parallel computers depends on two factors. These are the matrix by vector multiplications as well as the setup and application of the preconditioner.

Within the linearly–implicit extrapolation algorithms we get for discretized systems of PDE's Jacobians, which have a mainly banded structure, non–zero entries outside the main diagonal band refer to nonlocal couplings. Depending on the underlying discretization the matrices are frequently block tridiagonal or even may cover more than three diagonal blocks. Concerning the domain decomposition an effective parallel computation of the Jacobian a blockwise partitioning of the matrix is necessary, where each block corresponds to the equations in a subdomain assigned to one processor. This matrix storage scheme enables a very efficient parallel matrix by vector multiplication, requiring only preliminary updates of some external boundary values of the distributed vector x. These updates can be implemented highly effective as local exchanges between neighbouring processors.

Independent of the selected preconditioner the general problem remains, how accurate the linear systems should be solved. If we use approximations instead exact solutions we get a perturbed extrapolation table, which we will now analyze in some detail for the ODE case. We refer to the perturbation concept introduced by Bornemann [7] for linear and the extension by Lang and Walter [31] to nonlinear adaptive FEM methods. These authors searched for a relation between the accuracy of elliptic solvers and those due to the time discretization. We, however, are looking for a connection between the accuracy of the time discretization and those of the linear solver, therefore we use a somewhat different terminology.

As already mentioned we accept $T_{j,j}$ as approximation for $y(t_0 + H)$ using exact solutions of the linear systems, if $\|T_{j,j-1} - T_{j,j}\| \leq \varepsilon$ in an appropriate norm. Equivalently we require using approximate solutions of the linear systems, that $\|\tilde{T}_{j,j-1} - T_{j,j}\| \leq \varepsilon$, with $\tilde{T}_{j,k}$ as the approximations we obtain instead of $T_{j,k}$. Since $T_{j,j}$ is not known, the following estimate is obvious.

$$\|\tilde{T}_{j,j-1} - T_{j,j}\| \leq \|\tilde{T}_{j,j-1} - \tilde{T}_{j,j}\| + \|\tilde{T}_{j,j} - T_{j,j}\| \tag{11}$$

Both terms on the right–hand of (11) side now should be less than $\varepsilon/2$. This equation is the equivalent of equation (3.56) in [31]; the first term on the right–hand side is numerical availabe during the extrapolation, therefore we still need $\|\tilde{T}_{j,j} - T_{j,j}\|$. The terms $T_{j,k}$ are recursively defined by (5). Here we carry out the corresponding calculations only for $j = 1, 2$, using δ_{jk} for the errors made by the computation of $\tilde{T}_{j,k}$ and $\tilde{y}_{jl}$, the intermediate solutions. For $j = 1$ we simply obtain

$$\tilde{T}_{1,1} = y_0 + (I - HA)^{-1} H f(y_0) + \delta_{11} = T_{1,1} + \delta_{11} \; , \tag{12}$$

and for $j = 2$

$$\begin{aligned} \tilde{y}_{21} &= y_0 + (I - \frac{H}{2}A)^{-1} \frac{H}{2} f(y_0) + \delta_{21} \qquad = y_{21} + \delta_{21} \; , \\ \tilde{T}_{2,1} &= \tilde{y}_{21} + (I - \frac{H}{2}A)^{-1} \frac{H}{2} f(\tilde{y}_{21}) + \delta_{22} \; . \end{aligned} \tag{13}$$

With $T_{2,2} = 2\,T_{2,1} - T_{1,1}$ we can conclude, that

$$\begin{aligned} \|\tilde{T}_{1,1} - T_{1,1}\| &= \|\delta_{11}\| \; , \\ \|\tilde{T}_{2,2} - T_{2,2}\| &= \|2\delta_{21} + 2\delta_{22} - \delta_{11} + (I - \frac{H}{2}A)^{-1} H (f(\tilde{y}_{21}) - f(y_{21})\| \; . \end{aligned} \tag{14}$$

It seems difficult to proceed further without using a classical Lipschitz condition for f, which is not appropriate in the context of *stiff* ODEs, see e.g. DEUFLHARD [20].

Therefore we could use the results in [7], where for scalar linear PDEs the following connection between the required accuracies for $T_{j,k}$ in the extrapolation tableau and the space discretization error ε is derived

$$\delta_{jk} \leq \frac{\alpha_j^k}{j} \varepsilon \; , \tag{15}$$

and hope, that even in our case these estimates are satisfactory. The coefficients α_j^k/j depend only from the stepsize dividing sequence, their values decrease with increasing k and j. Consequently one has to solve the linear systems for higher j more accurately than for $j = 1$. For clarity we assume moreover that for a fixed j we solve all linear systems with a common accuracy, that means $\delta_j := \|\delta_{j1}\|, \ldots, \|\delta_{jj}\|$. As an rough approximation to the α_j^k we can use

$$\delta_1 \leq \frac{\varepsilon}{10} \; , \qquad \delta_2 \leq \frac{\varepsilon}{100} \; , \qquad \delta_j \leq \frac{\varepsilon}{1000} \; , \quad j = 3, \ldots \; . \tag{16}$$

These settings are of cause very pessimistic as could be expected due to the use of many triangle–inequalities. Indeed our practical tests have shown, that with

$$\delta_1 \leq \frac{\varepsilon}{10} \; , \qquad \delta_j \leq \frac{\varepsilon}{100} \; , \quad j = 2, \ldots \tag{17}$$

one always obtain solutions satifying the required accuracies and this values are used for all computations presented in this paper.[1]

Througout our investigations we used *Left* preconditioning. This is a natural choice, because we are not truly interested in the residuals, but in the errors, which of course are not available. Left preconditiong transforms the unpreconditioned residuals $r_k = b - Ax_k$ of the approximate solutions x_k in such a manner, that their norms should be of same magnitude as the errors e_k

$$\begin{aligned} P^{-1}r_k &= P^{-1}b - P^{-1}Ax \\ &\approx A^{-1}b - x_k \qquad = e_k \end{aligned} \tag{18}$$

This hope will be fulfilled if the preconditioner is really a "good" approximation for A. Indeed we are looking for such preconditioners, since within the extrapolation scheme one has to solve linear systems with several right–hand sides, therefore some expense for there computation may be worthwhile.

4.1 Block Jacobi and block Neumann preconditioners

The blockwise partition of the Jacobian leads to a quite effective Jacobian evaluation and matrix by vector multiplication. However, fundamental difficulties arise when applying standard preconditioners such as ILU or SSOR. The major bottleneck are the backsolves involving the triangular LU–factors and, for the ILU algorithm, even their computation. Since we use distributed matrices, the main loop in the sequential algorithm for solving triangular systems would be divided over the processors leading to very ineffective methods.

An alternative technique targeted more specifically at parallel environments is the block Jacobi preconditioner, or so–called additive Schwarz procedure. This simple approach uses complete or incomplete factorizations on the subdomains assigned to the processors, which can be computed and applied in parallel. Even if theoretically these factorizations do not need to exist if the matrices are not block diagonally dominant, in practice this never occurs because the subdomain matrices can be interpreted as the result of a discretization of smaller well–posed physical problems. Accordingly the change from A to P_{Jac} implies a decoupling of the original problem in these smaller ones. Formally the block Jacobi preconditioner is defined by $P_{Jac} = A_1 \oplus A_2 \oplus \cdots \oplus A_p$, p is the number of processors, see Figure 2.

We have implemented the block Jacobi preconditioner based on complete LU factorizations on the subdomains in the LIMEX code. As iterative solver we choosed GMRES [40], which is proven to be one of the most general applicable iterative methods. Nevertheless, we have made comparisons with other algorithms also, see §4.6. The initial

[1] If one, however, uses a classical Lipschitz condition as $LH < 1$, then one easily derives inequalities like $\|\tilde{T}_{2,2} - T_{2,2}\| \le \delta_1 + 5\delta_2$, $\|\tilde{T}_{3,3} - T_{3,3}\| \le \frac{1}{2}\delta_1 + 12\delta_2 + \frac{171}{8}\delta_3$, ... Herewith and again with (16) the conditions $\|\tilde{T}_{j,j-1} - T_{j,j-1}\| \le \varepsilon/2$ are satisfied as well.

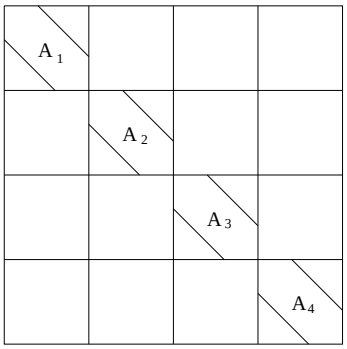


Figure 2: Block Jacobi preconditioner P_{Jac} of a banded matrix on four processors

guesses were always set to 0, this corresponds to the most obvious initial approximations $y_{k+1} = y_k$, see the code structure of LIMEX in §2. As already mentioned, we should use preconditioners which are good approximations for A. Accordingly we expect low iteration numbers and therefore we did not use the restart capabilites of the GMRES algorithm. As test problem we use in this chapter a system of 11 PDE's, which models the startup phase of an automobile catalytic converter. A detailed description of the problem can found in NOWAK [34] and EIGENBERGER AND NIEKEN [24]. This problem is characterized by strongly varying time–dependent chemical dynamics and stiffness. The speedups are always measured by comparison with the fastest sequential implementation using direct LU factorization. Therefore for some parallel variants we may obtain on one processor speedups less than 1. As usual, the performance results with the block Jacobi preconditioner are not encouraging, see Figure 3. One obtains only a speedup of about 2.3 with 16 processors, the number of iterations needed by the GMRES algorithm increases rapidly with the number of processors. Moreover, if one reduces the required accuracy of the preconditioned residuals, the solution exhibits spurious oscillations around the processor boundaries, which indicates that the coupling across the processor boundaries is not enough considered.

A natural extension of the block Jacobi approach is polynomial preconditioning. Hereby the preconditioner is constructed as a polynomial over A, usually of low degree, which approximates the inverse of A. The application of such preconditioners can be computed as a sequence of matrix by vector multiplications and is therefore quite effective using the good performance of matrix–vector products. To introduce polynomial preconditioners we define the matrix splitting

$$\omega A = D - (D - \omega A) \tag{19}$$

with the scalar relaxation parameter ω and the diagonal D of A. Then we get with $N = I - \omega D^{-1} A$

$$(\omega A)^{-1} \quad = \quad \left[D \left(I - (I - \omega D^{-1} A) \right) \right]^{-1}$$

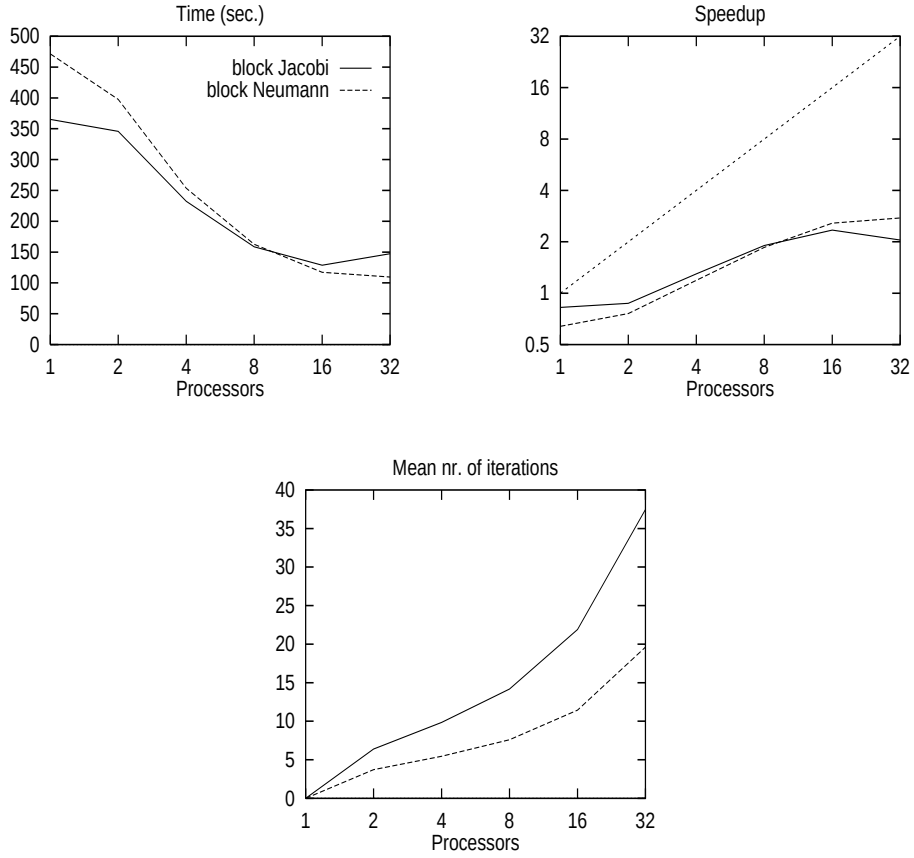


Figure 3: Performance results for the block Jacobi and Neumann preconditioner

$$\begin{aligned} &= (I-N)^{-1}D^{-1} \\ &\approx (I+N+N^2+\cdots+N^k)\,D^{-1} \end{aligned} \tag{20}$$

assuming that the spectral radius $\rho(N)$ satisfies $\rho(N) < 1$. Truncating this expansion at $k = 1$ leads to

$$(\omega A)^{-1} \approx (2I - \omega D^{-1}A)\,D^{-1}\ . \tag{21}$$

This preconditioner is improved by the usual replacement of D by P_{Jac}, we just used as preconditioner itself. Numerical experiences let to the choice $\omega = 1$, therefore we define the block Neumann preconditioner as

$$P_{Neu}^{-1} = (2I - P_{Jac}^{-1}A)\,P_{Jac}^{-1}\ . \tag{22}$$

This preconditioner was first introduced by Dubois et al. [23] in the solution of symmetric positive definite systems. Later da Cunha and Hopkins [15] provided evidence that this method is effective for non–symmetric systems as well. Obviously the application of P_{Neu} requires one more matrix by vector multiplication and one additional application of the block Jacobi preconditioner P_{Jac}.

The block Neumann preconditioner can also be defined through some other approaches. First one easily sees that the classical Richardson iteration

$$x_{k+1} = (I - D^{-1}A)\, x_k + D^{-1}b\,, \tag{23}$$

assuming $x_0 = 0$, produces as iterates x_k nothing else than the values of the truncated Neumann series. Thus the (block) Jacobi preconditioner is the result of one step, the (block) Neumann preconditioner the result of two steps of (23). The application of these preconditioners within an iterative method can therefore be seen as a nested iterative scheme, the inner iterations are now one or more steps of the Richardson algorithm.

Furthermore the approximations x_{2k} of the Neumann series expansion are also obtainable by a Newton algorithm applied to the function $f(x) = 1/x - a$ (Pan and Schreiber [36]). Then one gets for the vector–valued equivalent of f using $A_0 = D^{-1}$ as approximations for A^{-1}

$$A_{k+1} = (2I - A_k A)\, A_k\,. \tag{24}$$

This equation can be identified as a recursive definition for the preconditioners P_{2k} obtained by the Neumann series expansion, if one sets $P_{2k}^{-1} = A_k$. The different interpretations suggest a variety of modifications of our algorithms we have not yet fully investigated.

The results of applying the block Neumann preconditioner are not significantly better than the results of the block Jacobi method, see Figure 3, even if the number of iterations needed by the GMRES algorithm is now nearly halved compared to the block Jacobi method. Much more important is the increased robustness of the solutions, i.e. we obtain no more artificial oscillations near the processor boundaries. This evidently is a result of the coupling property of the preconditioner through the application of A within itself.

Truncating the series in (20) at higher k leads accordingly to higher order Neumann preconditioners, e.g. for $k = 2$ to

$$P_{Neu,2}^{-1} = \left[\, 3I - 3\, P_{Jac}^{-1} A + (P_{Jac}^{-1} A)^2 \,\right] P_{Jac}^{-1}\,. \tag{25}$$

The implementation of these higher approximations shows no further improvement. Even if the number of iterations further decreases, this is paid with an increased overhead for the application of the preconditioners. Therefore we did not study higher order polynomial preconditioners in more detail.

4.2 Explicit computation of the matrix product $P_{Jac}^{-1} A$

One common property of the block Jacobi and the block Neumann approach used in an iterative solver is the repeated application of the operator $P_{Jac}^{-1} A$. Therefore we analyze the structure of this product in more detail. Since we use *exact* LU factorizations on the subdomains, we can depict the matrix multiplication schematically as follows (we use the same terminology as in [1]):

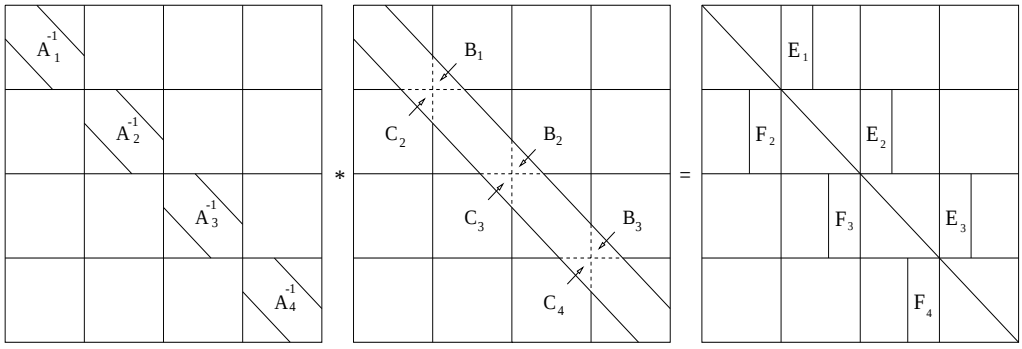


Figure 4: Schematic representation of the matrix product $P_{Jac}^{-1} A$ on 4 processors

In Figure 4 the E_i and F_i are defined by

$$E_i = A_i^{-1} B_i \qquad \text{and} \qquad F_i = A_i^{-1} C_i \; ; \tag{26}$$

the diagonal elements of $P_{Jac}^{-1} A$ are equal 1. We assume for clarity that the dimension n of A is a multiple of the number p of processors and A has m lower and upper diagonals. Then each of the matrices E_i and F_i covers m vectors of the length n/p. In the practical implementation of course one may use the block band structure of A. This further reduces the numbers of vectors in E_i and F_i. As an example, if A is block tridiagonal E_i, F_i consists of $(m+1)/2$ vectors. The structure of $P_{Jac}^{-1} A$ resembles to the Schur complement techniques. Indeed our algorithm can also be understood as a generalized Schur complement method combined with block Gaussian elimination but without labeling the interface nodes last.

The matrices E_i and F_i are computable completely in parallel. Once computed the application of the whole operator $P_{Jac}^{-1} A$ is reduced to some "small" matrix by vector multiplications, which are also fully parallelizable, provided the local updates of the right–hand side vector are already done.

Moreover the special structure of $P_{Jac}^{-1} A$ enables an estimation of the maximal number of iterations needed by the GMRES algorithm or other Krylov subspace methods. If we apply this matrix product to a vector v, then the vector $P_{Jac}^{-1} A\, v$ is obviously a linear combination of v and vectors from E_i and F_i and this is true also if $P_{Jac}^{-1} A$ is applied repeatedly. Therefore the Krylov subspaces

$$\mathcal{K}_k(P_{Jac}^{-1} A, v) = span\{v, P_{Jac}^{-1} A\, v, (P_{Jac}^{-1} A)^2\, v, \ldots, (P_{Jac}^{-1} A)^{k-1}\, v\} \tag{27}$$

have a maximal dimension of $2m(p-1)+1$, thus $\mathcal{K}_n$ as well. But since $\mathcal{K}_n$ contains the desired solution, this solution is already in $\mathcal{K}_{2m+1}$. This proves, that we need at most $2m(p-1)$ GMRES iterations with the block Jacobi or block Neumann preconditioner to obtain the true solution. This limit is for block tridiagonal matrices further reducible to $(m+1)(p-1)$. It is important, that these limits do *not* depend on the size n of the matrix, only linearly on the bandwidths and the number of processors. However, these upper bounds for the number of iterations should not be seen as realistic estimations, indeed we do not need exact solutions within the extrapolation scheme.

We note, that a similar statement is not possible for Krylov subspace methods associated with the transposed matrix. Even if again $\mathcal{K}_{2m+1} = \mathcal{K}_n$, we do not know, whether the solution is in $\mathcal{K}_n$. Therefore one merely obtains the result, that after $2m(p+1)$ iterations the Krylov spaces are "exhausted", perhaps then a restart will be necessary.

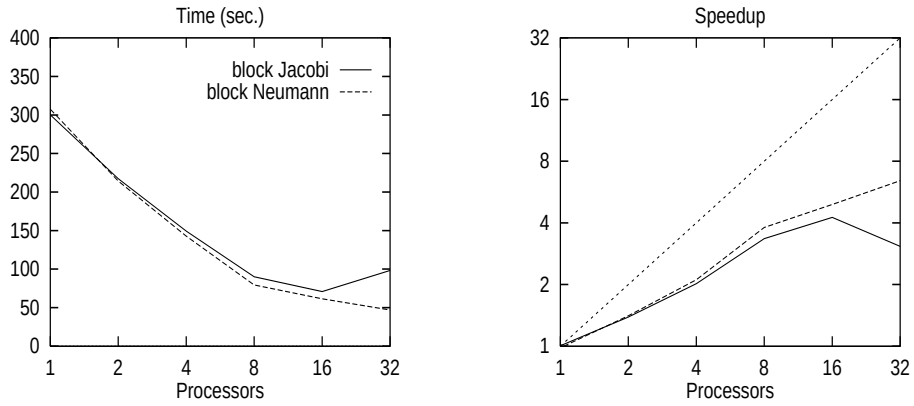


Figure 5: Performance results for the block Jacobi and Neumann preconditioner using explicit computation of $P_{Jac}^{-1} A$

The computation of the matrices E_i and F_i can be implemented very efficiently. The $2m$ vectors, respectively $m+1$ vectors for block tridiagonal matrices, together with the initial residual $P_{Jac}^{-1} A\, b$ are obtainable with a single call of a BLAS 2 subroutine. This is clearly illustrated by the performance results, see Figure 5. The number of iterations is identical to these in Figure 3 due to the algorithmic equivalence to the methods of the last section. We obtain now a speedup factor of 4 with 16 processors with the block Jacobi, and one of about 6.4 with 32 processors with the block Neumann method. For the latter algorithm the whole computing time is reduced about 50% by the explicit computation of $P_{Jac}^{-1} A$.

Even if the gain of performance is remarkable, there are also some drawbacks we should mention. First the advantages of the method are lost, if one works with *incomplete* factorizations on the subdomains. This could be necessary, if the subdomains itself are very large, and exact eliminations are too expensive, e.g. for higher dimensional

problems. Second we can not use the simplified algorithmic scheme explained in §2, obtained by replacing the fixed-point iteration by an iterative solver, e.g. GMRES, since then the product of the preconditioner matrix and A has not the simple algebraic structure displayed in Figure 4. Currently these drawbacks are outweigthed by the high performance of the iterative solvers using the explicit matrix product.

4.3 Overlapping domain decompositions

As already mentioned a reasonable preconditioner should suppress artificial effects of the domain decomposition. A powerful and well–known technique for this is to work with a slightly overlapping decomposition of the underlying domain, first introduced by Radicati and Robert [38], later investigated e.g. by Bjørstad and Widlund [5, 6], Cai et al. [12] and Gropp et al. [27]. Such methods define some gridpoints as overlap region or interface and apply the iterative solution algorithms on the subdomains as before, but the overlap region must be treated in a special manner. Usually when applying the preconditioner to a distributed vector, the values on the overlap region are averaged from the values computed by the local solvers. Other methods are the exchange of the overlapping data or the definition of artificial boundary conditions on the interfaces together with augmented matrices, sometimes called Schwarz enhanced matrices, see de Sturler [45]. The advantage of averaging is that the vectors used to iterate are the same on different processors. Averaging or exchanging follow every application of the block Jacobi preconditioner, thus

$$M_{av}\, P_{Jac}^{-1}\, A \qquad \text{or} \qquad M_{ex}\, P_{Jac}^{-1}\, A \tag{28}$$

replace the operator $P_{Jac}^{-1} A$ in both block preconditioners. The algorithmic realization of these operators needs no additional calls of communication routines, only the amount of data exchanged between neighbouring processors is increasing and depends on the size of the interface region.

Applying the approach of the overlapping domain decomposition gives a remarkable rise of efficiency especially for the block Jacobi preconditioner, see Figure 6. With a definition of 5 gridpoints as overlap region the computing times and the mean number of iterations for GMRES is reduced by approximately 50%. Even if a further increase of the overlap region reduces the iterations somewhat more, the increasing overhead for the solution of the linear systems inhibits a further speedup. Comparisons we have made between the averaging and exchanging variants yield better convergence properties for the averaging method, indeed the differences are not very significant. This is likewise observed by Lo and Saad [32]. The results are less dramatic for the block Neumann preconditioner, which itself does not ignore the coupling between the subdomains. One obtains a reduction for the global computing time of about 25% and for the iterations of about 50%.

From now on we further do not investigate the block Jacobi preconditioner which can not compete with the block Neumann approach.

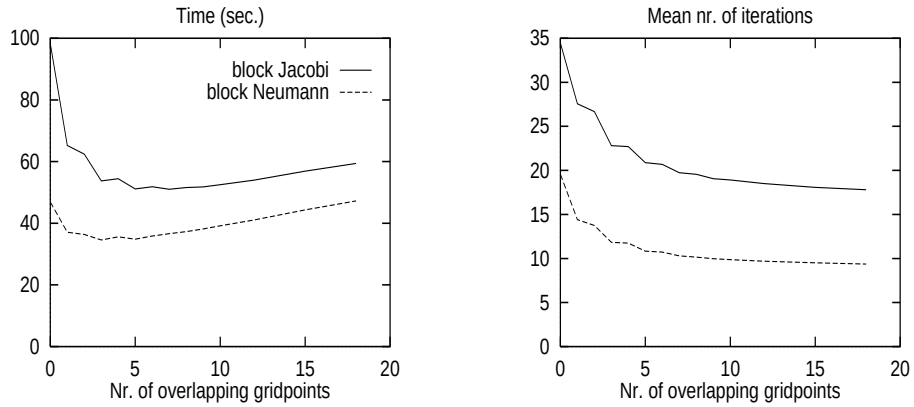


Figure 6: Performance results as a function of the size of the overlap region for the block Jacobi and Neumann preconditioner with 32 processors

4.4 Reduced system technique

In this section we will analyze in more detail the linear systems which are the result of the explicit formation of $P_{Jac}^{-1} A$ on p processors. Let E_i, F_i again be the submatrices introduced in § 4.2, n the size of the system and for simplicity m the upper and lower bandwith. Then we partition the matrices $E_2, \ldots, E_p$ and $F_1, \ldots, F_{p-1}$ into their first m, middle $n/p - 2m$, and last m rows. Similarly we divide E_1 into its first $n/p - m$ and last m and F_{p-1} into its first m and last $n/p - m$ rows and let the vectors x and b equivalently be partitioned. This is schematically depicted in Figure 7.

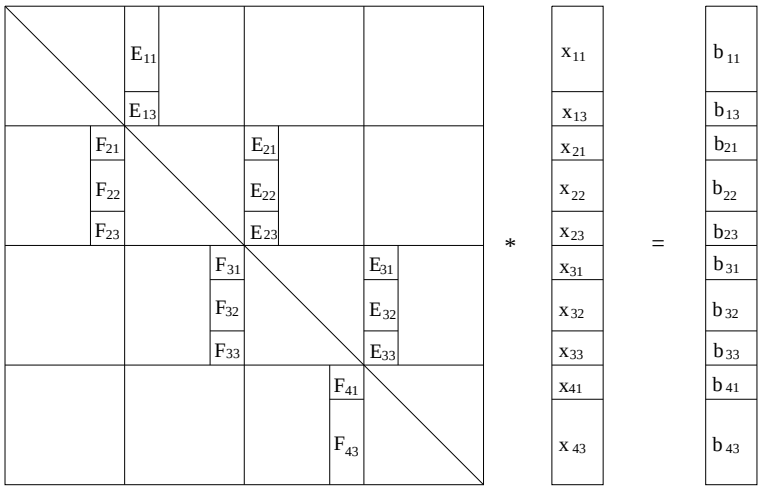


Figure 7: Partitioning of the linear system $P_{Jac}^{-1} A\, x = b$ on 4 processors

If we now rewrite the equations, we obviously split up the whole linear system as follows

$$\begin{array}{rcrcrcl} & & x_{13} & + & E_{13}x_{21} & = & b_{13} \\ F_{21}x_{13} & + & x_{21} & + & E_{21}x_{31} & = & b_{21} \\ F_{23}x_{1} & + & x_{23} & + & E_{23}x_{31} & = & b_{23} \\ & & & & & \vdots & \\ F_{p-1,3}x_{p-2,3} & + & x_{p-1,3} & + & E_{p-1,3}x_{p1} & = & b_{p-1,3} \\ F_{p1}x_{p-1,3} & + & x_{p1} & & & = & b_{p1}\ . \end{array} \tag{29}$$

This so–called *reduced system* is of order $2m(p-1)$ and therefore much smaller than the original system (assuming that $n \gg p$ and $n \gg m$). Considering the vectors x and b the equations operate only on the processor boundaries and can be solved completely independent of the remaining equations

$$\begin{array}{rcrcrcl} & & x_{11} & + & E_{11}x_{21} & = & b_{11} \\ F_{22}x_{13} & + & x_{22} & + & E_{22}x_{31} & = & b_{22} \\ F_{32}x_{23} & + & x_{32} & + & E_{32}x_{41} & = & b_{32} \\ & & & & & \vdots & \\ F_{p-1,2}x_{p-2,3} & + & x_{p-1,2} & + & E_{p-1,2}x_{p3} & = & b_{p-1,2} \\ F_{p3}x_{p-1,3} & + & x_{p3} & & & = & b_{p3}\ . \end{array} \tag{30}$$

This system with the unknowns $x_{11}, x_{22}, x_{32}, \ldots, x_{p-1,2}, x_{p3}$ can be solved afterwards using the solutions of the reduced system by some inexpensive matrix by vector multiplications.

The reduced system itself is a distributed block tridiagonal matrix with blocks of order $2m$, the possible combination with an overlapping domain decomposition leads to slightly greater blocks. To solve this system we use again iterative methods, but now all vectors used by these algorithms, e.g. the distributed Krylov vectors in GMRES, are very short, their length is only of the order of the reduced system $2m(p-1)$ instead of n. This is the main advantage of the method, because the number of floating point operations is remarkably reduced. The convergence properties of an iterative method applied to the full matrix–product or the reduced system should be very similar. The eigenvalues of the reduced system are eigenvalues of $P_{Jac}^{-1}A$ as well, and this matrix itself has the single additional eigenvalue 1 with multiplicity $n-2m(p-1)$. Considering the size of the reduced system we see again, that at most $2m(p-1)$ GMRES iterations are needed to solve the whole linear system.

Since we have now completely developed the methods implemented in our currently most efficient parallel LIMEX version, we summarize its algorithmic structure as in §2 for the sequential counterparts.

Compute the distributed Jacobian $A = (f(y_0) - By')_y$ *in parallel on an overlapping domain decomposition*

for $j = 1, \ldots, j_{max}$ *while convergence criterion not satisfied*

$h_j = H/n_j$

Compute in parallel P_{Jac}^{-1} *(Fig. 2) of* $B(y_0) - h_j A$

Compute in parallel the matrices E_i, F_i *and the vector* $P_{Jac}^{-1} h_j f(y_0)$

Solve iteratively $y_1 = y_0 + (B(y_0) - h_j A)^{-1} h_j f(y_0)$
using the preconditioner P_{Neu} *(22) for the reduced system*

Compute the whole solution of the linear system

for $k = 1, \ldots, j-1$

Solve iteratively $y_{k+1} = y_k + (B(y_k) - h_j A)^{-1} h_j f(y_k)$
using the iterative algorithm (9) and the preconditioner P_{Neu}
for the reduced system

Compute the whole solution of the linear system

$T_{j,1} = y_j$

if $j > 1$ *compute* $T_{j,j}$ *and check convergence*

$y_{new} = T_{j,j}$

There are some techniques related to the reduced system method which we should briefly consider. We believe, that the inherent equivalence between these aproaches is not commonly recognized. First the reduced system method is introduced in detail in [1] to develop parallel direct solution methods for banded systems. Furthermore Brakkee et al. [8, 9] investigated in the context of the incompressible 2D Navier–Stokes equation domain decomposition methods and derived a so–called interface–equation in connection with exact subdomain solutions, which is nearly completely equivalent to the reduced system. In a somewhat more general manner, Bramley and Meñkov [10] have examined low rank off–diagonal block preconditioners. They studied approximations for a sparse matrix which can be written as $B = C + UV^T$, where U and V matrices composed of only a "few" vectors. Applying the Shermann–

Morrison–Woodbury formula [26] one obtains

$$B^{-1} = C^{-1} - C^{-1}U(I + V^T C^{-1} U)^{-1} V^T C^{-1} \ . \tag{31}$$

Now $I + V^T C^{-1} U$ has an order equal to the rank of U and V. Therefore, if B is easily factorizable it remains only the treatment of this "small" matrix. Our approch can be embedded in their terminology, one actually easily constructs matrices U and V, each composed of $2m(p-1)$ vectors, with the property $P_{Jac}^{-1} A = I + UV^T$. Then with $C = I$ indeed $I + V^T U$ is exactly the reduced system.

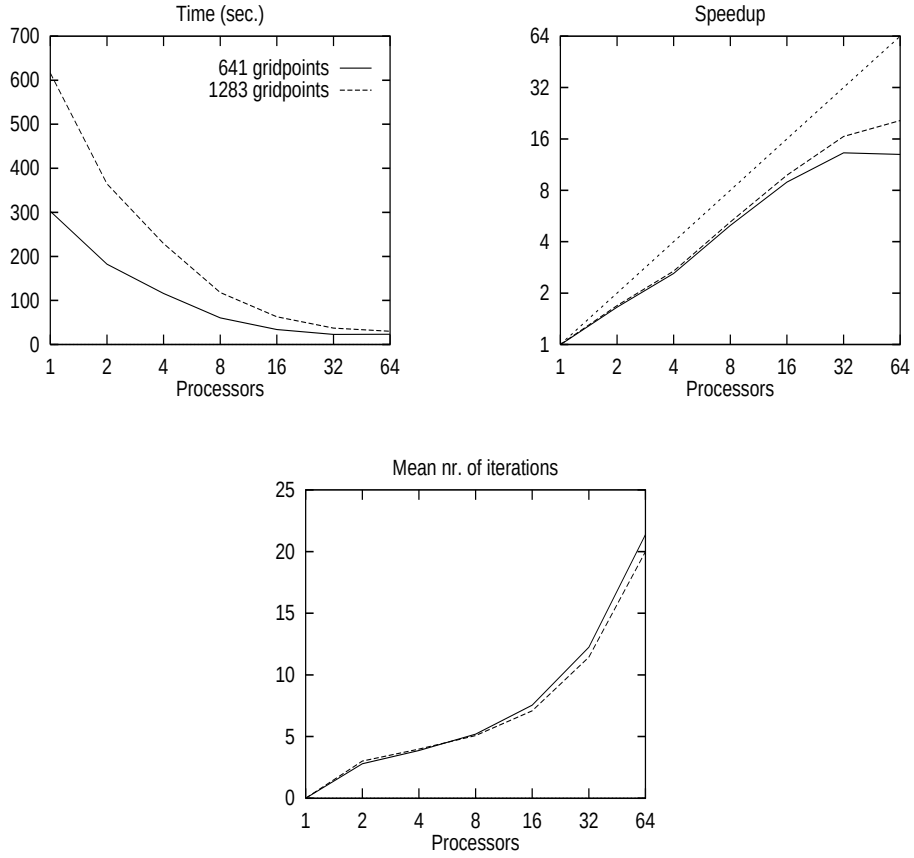


Figure 8: Performance results with the reduced system technique on two gridsizes

Applying the described reduced system technique combined with an overlapping domain decomposition we once more obtain a substantial gain of performance, see Figure 8. Because the iterations now are very cheap, we get the minimal computing time, if the overlap region consists of a single grid point. At a first sight, the algorithm exhibits scalability only up to 32 processors. But we have to consider that our test problem with 641 gridpoints, which leads to linear systems with 7051 unknowns, is

only of medium size and with 64 processors each processor has to manage only about 10 gridpoints. If we simply enhance the grid size by a factor 2 we obtain a speedup of 20.6 with 64 processors, see Figure 8.

The overall scalability of the reduced system technique together with an overlapping domain decomposition and block Neumann preconditioning will be clearly shown in §5 applying the method to a large difficult problem from chemical engineering.

The reduced system technique suggests once more an attempt to construct a direct solver, but now only for this much smaller system. This construction, which is equivalent to the direct parallel band solver proposed by DONGARRA AND SAMEH [22], consists of the following steps.

Factorization phase:

1. Parallel computation of $P_{Jac}^{-1} A$.
2. Distribution of *all* matrices E_i and F_i over *all* processors.
3. Direct factorization of the reduced system on *all* processors by Gaussian elimination.

Solution phase:

1. Collection of *all* boundary regions of the right–hand side vector b on *all* processors.
2. Direct solution of the reduced system on *all* processors, this gives the solution x on the boundary regions.
3. Parallel solution of the remaining unknowns between the processor boundaries.

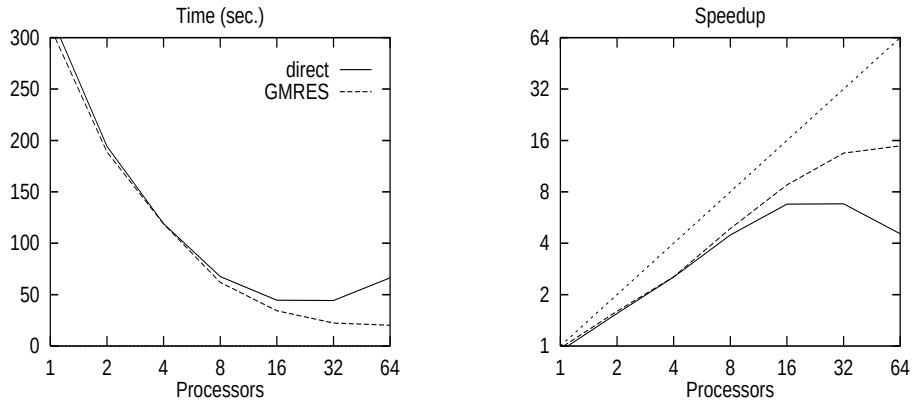


Figure 9: Direct solution of the reduced system vs. iterative solution

Even if this direct solution method cooperates with the domain decomposition, it is obvious that a large amount of non–local communication will be needed, in the factorization phase as well as in the solution phase. This severe drawback is clearly illustrated by Figure 9, where the performance is compared to our iterative solution method. Although the amount of data, which has to be distributed over all processors, is not very large for the medium-sized test problem and despite the high communication bandwidth of the Cray T3D, the direct method is competitive only up to 16 processors. For larger problems the differences become much more significant.

4.5 Comparison of some iterative algorithms

Until now we have developed our methods using the GMRES algorithm without restarts. In this section we will make some comparisons to other iterative solvers. At first we have implemented the full orthogonalization method (FOM) or Arnoldi algorithm [41]. This method is likewise GMRES a projection method, but does not guarantee a monotone decrease of the residual norm. There are some relations between both methods, see Cullum and Greenbaum [14], accordingly these algorithms should be, from a practical viewpoint, nearly equivalent. If one uses a robust and efficient implementation, especially a recursive update of the LU factorization of the arising Hessenberg matrices, this assertion is even in the parallel environment clearly fulfilled, see Figure 10. We found no significant differences in respect to computing times or iteration numbers.

An other widely used iterative method is BICGSTAB [48]. It does not incorporate an orthogonalization process, but for any new approximate solution one needs two matrix by vector multiplications and applications of the preconditioner. Somewhat surprising this method is likewise just as effective as GMRES within our algorithms. Although the number of iterations is substantially smaller, by the more expensive iterations a possible gain of performance is nearly exactly outweighted.

The parallelization of GMRES has been studied by several authors. Some of them consider the basic GMRES iteration with the Arnoldi process, e.g. in [44, 4, 15]. Xu et al. [49] have proposed a variant called α–GMRES, which consists of a multilevel iterative scheme; the system formed at each step of the outer iteration is then solved by GMRES. Hybrid algorithms combining a GMRES basis with a Chebyshev basis have been investigated by Joubert and Carey [28]. Another method is to define a new basis, e.g. a Newton basis, of the Krylov subspaces, for example in [2, 13, 46, 29]. The common aim of all approaches is to avoid the large number of vector–vector operations and global communications present in the Arnoldi orthogonalization process.

Many of the proposed algorithmic variants use groups of vectors to reduce the number of global reduction routines. In our case (≤ 64 processors) we compute in average only 20 Krylov vectors and such approaches would be of very limited value for our parallel implementation, especially as the these methods decrease the *sequential* convergence

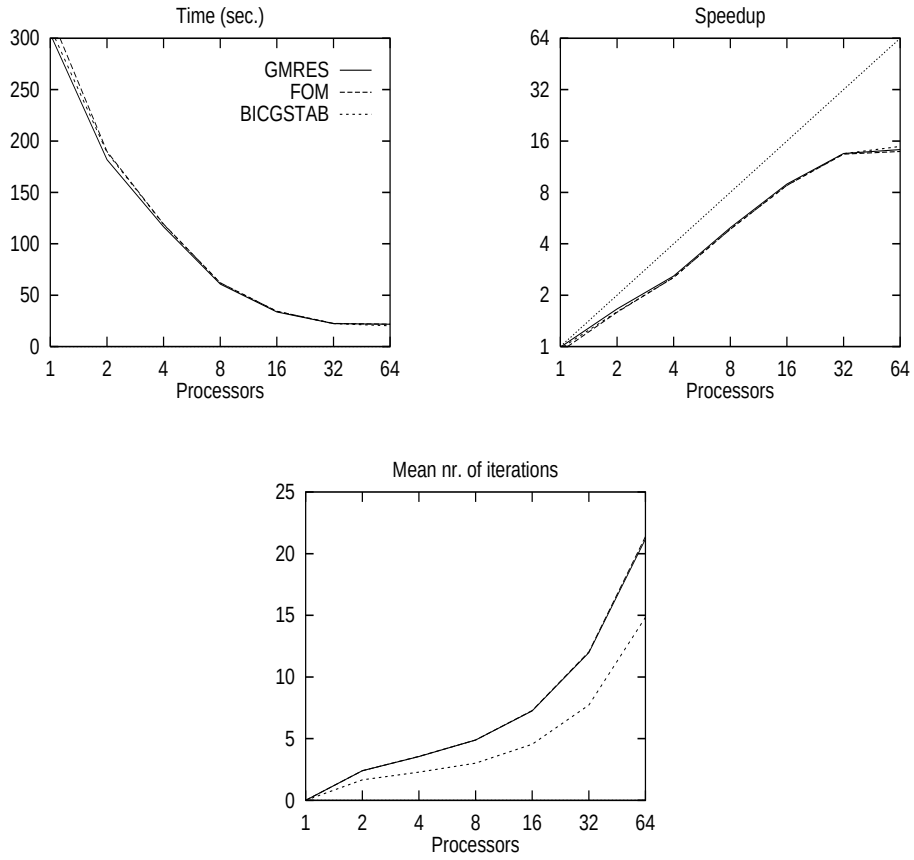


Figure 10: Performance results GMRES vs. FOM and BICGSTAB

speed. The same argument excludes algorithms with different bases.

As already mentioned, one of the potential bottlenecks in a parallel GMRES implementation is the orthogonalization process. In a sequential implementation this is typically carried out by the so–called modified Gram-Schmidt orthogonalization procedure. Its algorithmic kernel is the following loop.

$$
\begin{aligned}
&\textit{for } j = 1, \ldots, k \\
&\qquad h = (v_j, v_{k+1}) \\
&\qquad v_{k+1} = v_{k+1} - h\, v_i
\end{aligned}
$$

which orthogonalizes v_{k+1} against j already orthogonal vectors $v_1, \ldots, v_k$. Each inner product in the loop requires global communication and stands for one synchronization point. This gives k synchronization points and therefore the whole loop must be done in sequence. A well known remedy is to go back to the classical Gram–Schmidt procedure, which is sketched next.

$$
\begin{aligned}
&\textit{for } j = 1, \ldots, k \\
&\qquad h_j = (v_j, v_{k+1}) \\
&\textit{for } j = 1, \ldots, k \\
&\qquad v_{k+1} = v_{k+1} - h_j \, v_j
\end{aligned}
$$

Now the first loop can be computed completely in parallel. After that we can collect the inner products all at once. This yields the only synchronization point of the procedure. Due to its inherent numerical instability the classical Gram–Schmidt procedure tends to perform poorly for larger Krylov subspace sizes. Nevertheless we have implemented this method in our parallel GMRES algorithm, the performance results are shown in Figure 11. The results clearly indicate, that there is no significant difference

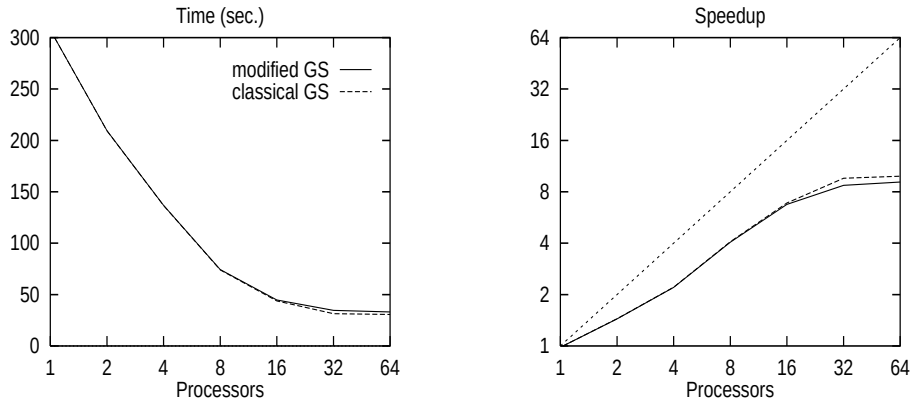


Figure 11: Performance results with GMRES and classical vs. modified Gram–Schmidt

between both approaches. The classical algorithm is slightly superior, and we get no problems with numerical instabilities, indeed the iteration numbers are always the same. It is therefore obvious, that we can trust the classical Gram–Schmidt procedure due to the high quality of our preconditioner which needs only small Krylov subspaces. But otherwise the disadvantage using the modified Gram–Schmidt can be nearly neglected, due to the very high communication performance of the Cray T3D. The latter conclusion would be false, if we would use a loosely coupled cluster of workstations instead of a tightly coupled massively parallel computer. As an example, Lo and Saad [32] reported on such systems differences of sometimes more than 50% between the overall performance of the classical respectively modified procedure.

We summarize that we need no special strategies to adopt the GMRES algorithm on our parallel environment. This statement is likewise true for the FOM– and the BICGSTAB–method.

4.6 Reusing of Krylov subspaces

Even if we do not need to look for a special parallel adaptation of our iterative solvers we should consider, that we solve within the extrapolation scheme *many* linear systems, some of them with the same matrices, some of them with matrices different only in the diagonal. This situation suggests to reuse somehow the work done in the preceeding steps.

The most obvious way to solve many linear systems with a single coefficient matrix and multiple right–hand sides is to use block generalizations of iterative solvers, see e.g. in SIMONCINI AND GALLOPOULOS [47]. Since in the context of the extrapolation the right–hand sides are not simultaneously available, such methods can not be used. The only way to profit from preceeding solver calls is therefore to reuse the already constructed Krylov spaces. To introduce our approaches we must refer to some algorithmic details of GMRES.

Assuming $Ax = b$ has to be solved by GMRES with $x_0 = 0$, this method constructs successively the Krylov subspaces $\mathcal{K}_k = span\{b, Ab, \ldots, A^{k-1}b\}$. If V_k is a basis of $\mathcal{K}_k$, then the minimal residual approximation in $\mathcal{K}_k$ is given by

$$x_k = V_k \left[(AV_k)^T AV_k \right]^{-1} (AV_k)^T b \; . \tag{32}$$

Defining the $(k+1) \times k$ Hessenberg matrix $\bar{H}_k$ by the relation $AV_k = V_{k+1}\bar{H}_k$ one obtains with $\beta = \|b\|$

$$\begin{aligned} x_k &= V_k \left[\bar{H}_k^T \bar{H}_k \right]^{-1} \bar{H}_k^T V_{k+1}^T \, b \\ &= V_K \left[\bar{H}_k^T \bar{H}_k \right]^{-1} \bar{H}_k^T \beta e_1 \qquad = V_k \, y_k \; , \end{aligned} \tag{33}$$

where y_k minimizes $\|\beta e_1 - \bar{H}_k y\|$ over $y \in \mathbb{R}^k$. This minimizer is computed via successive QR-decompositions of $\bar{H}_k$ using Givens rotations.

Let us now change from the single equation $Ax = b$ to the set of preconditioned linear equations $P^{-1}(B_h - hA) = P^{-1}b$ wich are to solve for different values of h and b, P is in general dependent from h. We will develop six methods, which use the already constructed Krylov subspaces and/or preconditioners in a different manner. First for completeness:

1. Solve every linear system independently with GMRES.

This is of cause exactly our hitherto used approach. An obvious simplification could be:

2. Use one common preconditioner for all linear systems within one basic step.

The most obvious choice for a h–independent preconditioner is surely $P = P_{H/2}$. Then we need only once the computation of a preconditioner for one whole integration step. The disadvantage is clearly that one can not use the reduced system technique for $h \neq H/2$, although we expect slower convergence exactly for this case.

Now we attempt to reuse the Krylov subspaces constructed in preceeding steps, first for one fixed h. Suppose we have already solved approximately one linear system with the coefficient matrix $B_h - hA$. The best approximation $\tilde{x} \in V_k$ for the solution of the same linear system with a *new* right–hand side b is then evidently given by

$$\tilde{x} = V_k \left[\bar{H}_k^T \bar{H}_k\right]^{-1} \bar{H}_k^T V_{k+1}^T P^{-1} b \; , \tag{34}$$

but the equation $V_{k+1}^T P^{-1} b = \|P^{-1}b\| e_1$ remains not valid, since the normalized vector $P^{-1}b$ is not equal to v_1. Therefore we must explicitly evaluate the formula $V_{k+1}^T P^{-1} b$. This can be implemented as a completely distributed matrix by vector multiplication, recognizing that the boundary regions of the Krylov vectors v_k are already exchanged during the GMRES iteration. After that one uses effectively the already computed QR decomposition of $\bar{H}_k$ to obtain $\tilde{x}$. Even if $\tilde{x}$ is a satisfactory approximate solution, we have to compute its preconditioned residual to be sure. If $\tilde{x}$ can not be accepted, we use $\tilde{x}$ as an initial guess for a new GMRES iteration. The last two steps are yet the possible drawback of the next two methods we present.

3. For each h take the in the first step computed Krylov subspace to compute approximate solutions. If the residuals are not less than the prescribed tolerance, start new GMRES processes.

4. As in 3., but use the h-independent preconditioner $P_{H/2}$.

A further advanced approach would be, even for different h, to look for approximate solutions in one common Krylov subspace. Assume we could dispose of a Krylov subspace $\mathcal{K}_k$ constructed by help of a stepsize h_0. Then the best approximate solution $\tilde{x} \in V_k$ for $P^{-1}(B_h - hA)x = P^{-1}b$ would be

$$\begin{aligned}
\tilde{x} &= V_k \left[\bar{H}_k^T V_{k+1}^T P^{-1}(B_h - hA)V_k\right]^{-1} \bar{H}_k^T V_{k+1}^T P^{-1} b \\
&= V_k \left[\bar{H}_k^T V_{k+1}^T P^{-1} \frac{h}{h_0}(B_{h_0} - h_0 A)V_k + \bar{H}_k^T V_{k+1}^T P^{-1}(B_h - \frac{h}{h_0}B_0)V_k\right]^{-1} \bar{H}_k^T V_{k+1}^T P^{-1} b \\
&= V_k \left[\frac{h}{h_0}\bar{H}_k^T \bar{H}_k + \bar{H}_k^T V_{k+1}^T P^{-1}(B_h - \frac{h}{h_0})B_{h_0})V_k\right]^{-1} \bar{H}_k^T V_{k+1}^T P^{-1} b \; .
\end{aligned} \tag{35}$$

A numerical evaluation of the last expression is not practicable, as long as we furthermore do not assume, that the underlying GMRES iteration has resulted in a very accurate approximate solution. Then, if $V_{k+1} = \{v_1, \ldots, v_{k+1}\}$, we have $v_{k+1} \approx 0$ and

for the matrix element $h_{k+1,k}$ from $\bar{H}_k$ consequently $h_{k+1,k} \approx 0$ and obtain after some manipulations

$$\tilde{x} \approx V_k \left[\frac{h}{h_0} H_k + V_k^T P^{-1} (B_h - \frac{h}{h_0} B_{h_0}) V_k \right]^{-1} V_k^T P^{-1} b \,. \tag{36}$$

with H_k the $k \times k$ Hessenberg matrix, which results from $\bar{H}_k$ by deleting the last column. Again the expression $V_k^T P^{-1} b$ is computable in parallel, but the evaluation of $V_k^T P^{-1}(B_h - \frac{h}{h_0} B_{h_0}) V_k$ needs k^2 global simultaneous inner products. The resulting matrix in (36) is a full $k \times k$ system and must be factorized by Gaussian elimination. As before the last steps are to check the residual and if needed the setup of a new GMRES iteration. Apparently these last two methods are algorithmic more expensive:

5. For an appropriate h_0 compute the Krylov subspace V_k and the Hessenberg matrix H_k. For all other linear systems take these to compute approximate solutions. If the residuals are not less than the prescribed tolerance, start new GMRES processes.

6. As in 5., but use an h-independent preconditioner.

We choosed $h_0 = H$, since the linear system $B_0 - HA$ appears as the first and produces in general Krylov subspaces with the largest dimension, i.e. it needs the most GMRES iterations.

The implementation of the enumerated approaches yielded in summary no advantage. All algorithmic variants using one common preconditioner (methods 2, 4 and 6) can not compete with our standard version (method 1). Computing time and the number of GMRES iterations increased remarkably. For the methods with an h-dependent preconditioner the results were not so discouraging, but we got no systematic gain of performance. Especially method 3 behaves very similar to method 1, method 5 is about 10% slower. An analysis of the convergence behaviour shows, that almost never the first Krylov subspace contains satisfactory approximations for the following right–hand sides. Thus each new b, however, needs again an own GMRES call within the number of necessary iterations decreases in the mean only about 20–30%. But due to the cheap iterations with the reduced system technique there is no overall measurable speedup.

This situation could be changed for higher dimensional problems, where the direct subdomain solutions may be too expensive and one has to work with preconditioners which need Krylov subspace of larger dimensions. Then the methods developed in this section should get our renewed interest. We should note that all results of this section may be dependent of the properties of a specific application, as e.g. their stiffness, which will finally be illustrated.

The fact, that only rarely a common Krylov subspace containing solutions for different right–hand sides exists, is indeed related to the stiffness of the underlying system of

ODEs. For simplicity we set $B = I$ and neglect the preconditioner. Assume that for an iterative method, for clarity we choose FOM, a common Krylov subspace with the basis V_k exists, which is produced by all iterative solutions of linear systems arising in one extrapolation step. Thus for all corresponding right–hand sides b we have analogously to (32)

$$(I - hA)^{-1}b = V_k \left[V_k^T (I - hA) V_k\right]^{-1} V_k^T b \; . \tag{37}$$

Now define the matrix $\tilde{A} = V_k V_k^T A V_k V_k^T$, which represents A only within the subspace V_k. Then one easily sees that

$$(I - h\tilde{A})^{-1} = V_k \left[V_k^T (I - hA) V_k\right]^{-1} V_k^T + I - V_k V_k^T \; . \tag{38}$$

Considering $b \in V_k$, we can conclude

$$(I - hA)^{-1}b = (I - h\tilde{A})^{-1}b \; . \tag{39}$$

Thus one could perform the whole extrapolation step using $\tilde{A}$ instead of A. With (38) we obtain therefore for the linearly–implicit Euler discretization

$$y_{k+1} = y_k + hf(y_k) + V_k \left[V_k^T (I - hA) V_k\right]^{-1} V_k^T hf(y_k) \; . \tag{40}$$

This equation can obviously interpreted as an application of the linearly–implicit Euler only within V_k and the *explicit* Euler, $y_{k+1} = y_k + hf(y_k)$, outside from V_k. For this reason a Krylov subspace of low dimension containing the solutions arising during one integration step would reflect an only mildly stiff behaviour of the ODE system, because "most" of the discretization could be done explicitely.

5 Parallel implementation issues

We implemented our algorithms on a Cray T3D with 256 processors, each with 64 MB memory. The codes are written in a hostless manner. Each processor is assigned to one subdomain and the information pertaining the interior of the subdomain is uniquely owned by that processor and is not availabe to any other processor except by explicit message passing. Order and stepsize control are done by all processors simultaneously, the CPU-time spent on it is negligible, likewise the QR decompositions within GMRES.

Except our first attempts with parallel direct solvers, where we used the BLACS library [21], all message passing calls are implemented through the shared memory access communication routines (SHMEM library) from Cray Research Inc. [3]. We wrote a small library containing only two types of routines: global reductions needed in inner products as an example, and local exchange operations between neighbouring processors. These local exchanges can be coded synchronous, then they act as synchronization point, or asynchronous, requiring some more buffering to be safe. We did not found large differences in respect to the overall computing times between both approaches. Up to 32 processors the synchronous exchanges are slightly faster, from 64 processors the asynchronous ones. Both variants of the local exchange routines use only the fastest point to point communication routine `shmem_put`. Furthermore we needed, only for the direct parallel solver of the reduced system, a global collector routine and for the distribution of input data a global broadcasting routine.

Because the codes itself never calls directly any communication function the program could be very easily switched to any other message passing library, especially MPI, changing only the routines within our library.

5.1 A heuristic scalability analysis

We conclude this section with an analysis of the scalability of our algorithms, but we do not count floating point operations or compute serial or parallel complexities. This approach may be only of very limited value for a whole application code, since it does not refer to the efficiency of assemby coded linear algebra kernels or to the latencies and communication performance of a specific parallel machine. Instead of that we have measured the computing times of the main parts of our program to demonstrate there different scalability potential, see Figure 12. [2]

[2] In Figure 12, the maxima in the curves for local exchanges and global reductions at 4 resp. 2 processors are explainable with wait times within the communication routines due to load imbalancies using only a few processors.

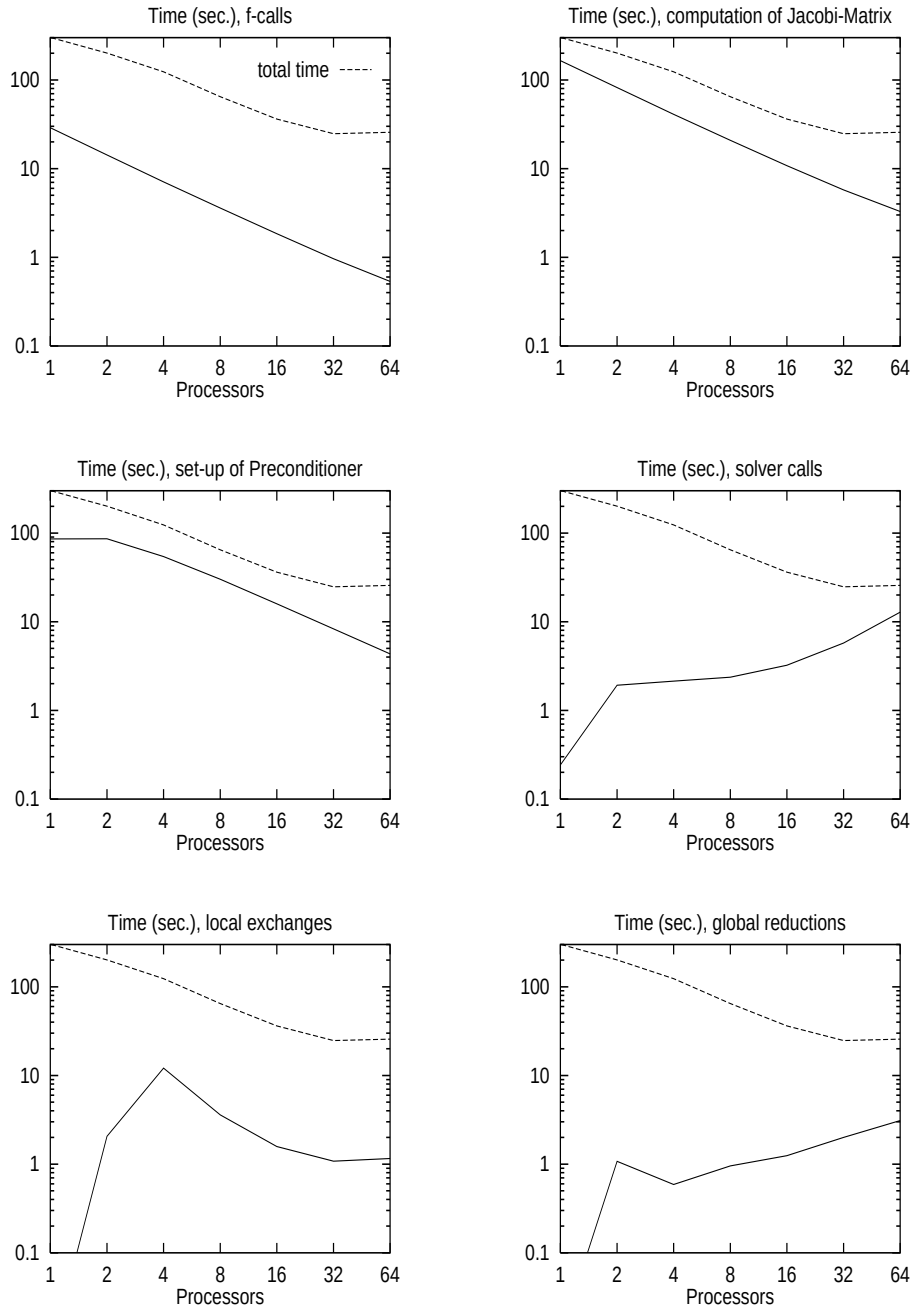


Figure 12: Scalability of the main parts in the parallel LIMEX implementation

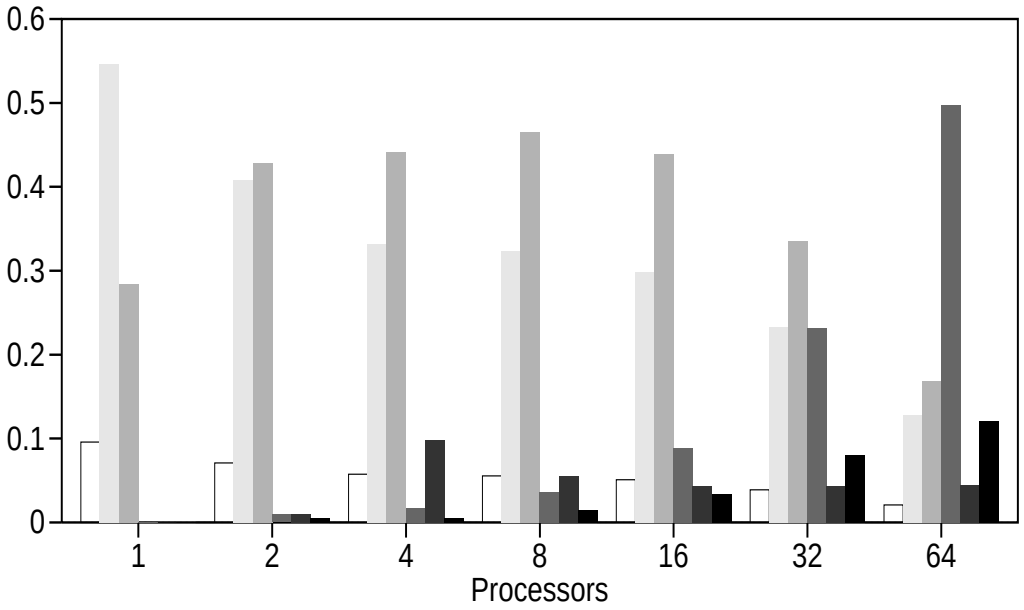


Figure 13: Relative contributions to the computing time

It is obvious, that the application specific parts, i.e. the calls of the right hand side f within the extrapolation and the evaluation of the Jacobian scales very well, likewise even the computation of the preconditioner, which can be done almost completely in parallel. Clearly an ideal scalability is prevented by the communication calls and in particular by the iterative solver due to the increasing number of iterations. The changing computational weights are ilustrated in Figure 13 as relative contributions of the 6 main parts of Figure 12 to the computing time. Indeed with 64 processors the solver calls predominate the whole computing time, whereas the application specific parts become more and more insignificant. This processor number, however, is in no way a general limit for the scalability of our methods, but only for the medium–sized test problem we selected. For much larger examples we can expect scalability even for higher processor numbers.

We should remark that the parallel efficiency of our codes strongly depends on the speed of local exchanges and global reductions, the latter mostly only for one single inner product, whereas other global communication or transfer between not neighbouring processors almost never occur. Indeed the former communication types are used permanently, in the extrapolation scheme as well during the computation of approximate solutions by an iterative solver. Therefore the tightly coupled Cray T3D is an ideal target machine for parallel extrapolation codes.

6 Results for an aerosol formation problem

In this section we will demonstrate the performance and scalability of our approaches when applied to a large and numerical difficult problem from chemical engineering. The aim of this application project is the development of a general tool to simulate stationary and instationary processes in chemical equipments including aerosol formation. The mathematical models include the description of the formation and growth of fog droplets by heterogeneous condensation of mixtures, supersaturation in the gas phase and coagulation processes and has been successfully applied e.g. to absorption processes for hydrochloric acid in industrial exhaust purification plants. We refer to Schaber and Körber [42, 43] for a detailed description.

The model equations for the involved thermodynamical processes and for the aerosol growth lead to a system of coupled hyperbolic and parabolic PDEs which has to be solved together with the nonlinear equations of heat and mass transfer as well as for the volume, mass and energy balances. The space discretization of the PDEs results in a set of differential and algebraic equations, which can be integrated by the extrapolation code LIMEX.

The modelling of the distribution of droplet sizes by a discrete set of classes requires the rebuilding of the whole system of PDEs after every time–step. Thus the number of PDEs *and* gridpoints and therefore the size of the linear systems regularly changes after each integration step. This type of problem can be efficiently solved with one–step algorithms as the extrapolation method, but not with standard multistep methods like BDF. We remark that due to the spatial changing distribution of droplets the number of ODEs assigned to each gridpoint are in general very different, this can lead to severe load balancing problems, if the actual grid is not appropriate partitioned. Furthermore the underlying upwind–discretization results in structural non–symmetric tetra–diagonal linear systems.

Regardless these difficulties, all algorithmic strategies we developed are applicable without any principial changes. Within the program package, which simulates the gas purification, the LIMEX code is only one of many subroutines, even the most time consuming one. We have made no attempts to parallelize the whole package. In order to be consistent, we have therefore after each integration step to distribute the solution vector over all processors.

We present in Figure (14)–(17) the performance results for three droplet size distributions, which stand for three different problem sizes. Hereby the mean gridsize, averaged over the whole integration is about 100, the size of the linear systems is about 3000 resp. 5000 or 8000. The block sizes fluctuate between 10 and 19, resp. 10 and 25 or 10 and 46. The overlap region in every computation consists of one gridpoint.

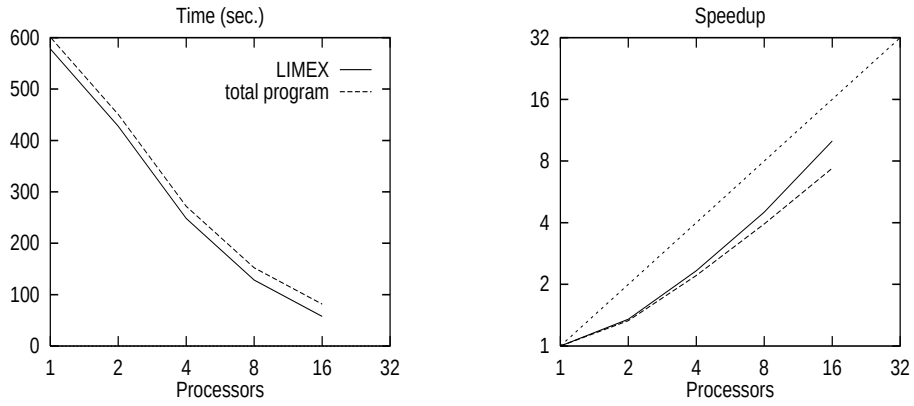


Figure 14: Performance results for the aerosol formation problem with 3 droplet sizes

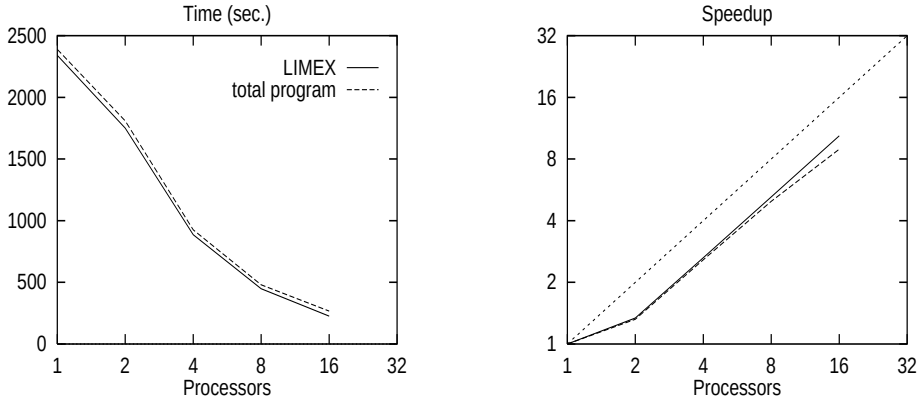


Figure 15: Performance results for the aerosol formation problem with 7 droplet sizes

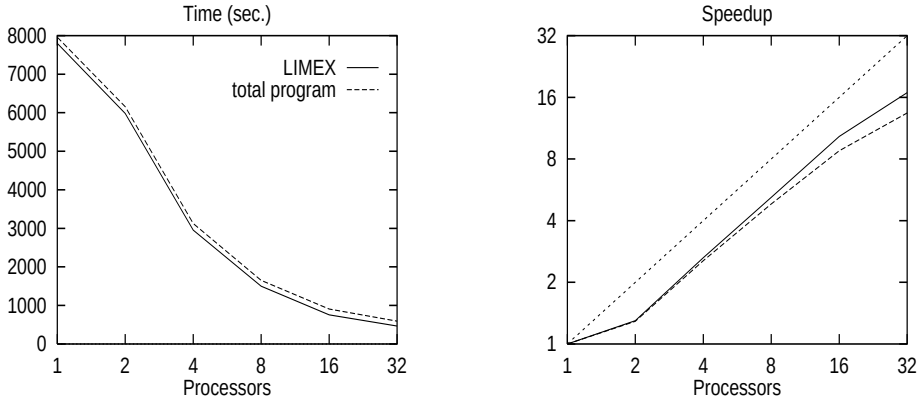


Figure 16: Performance results for the aerosol formation problem with 12 droplet sizes

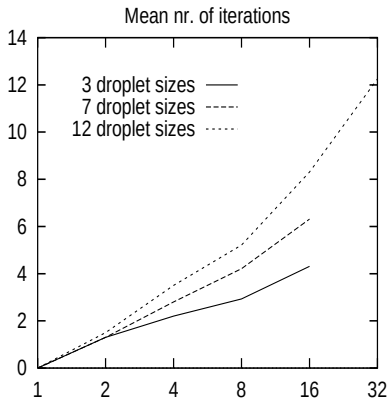


Figure 17: Mean iteration numbers for different droplet classes

The results clearly show the value of the parallel extrapolation method within application specific codes for the integration of large systems of DAEs. We obtain maximal speedups between 10.1 for the smallest problem with 16 processors and 16.9 for the largest with 32 processors. Even if the number of processors is still limited by the only medium–sized grids, the attainable speedup already enables the computation of much more complex problems. So effective parallel codes as presented in this paper, may be a powerful tool to allow realistic simulation of aerosol formation and growth in whole chemical equipments. Indeed even the aerosol problem we used here acts only as a small example.

7 Conclusion

We have investigated stategies to parallelize linearly–implicit extrapolation codes for the integration of large systems of ODEs and DAEs. We found that from many possible approaches one obtains best performances from a combination of a slightly overlapping domain decomposition together with a polynomial block Neumann preconditioner and the application of the reduced system technique. Since within the extrapolation algorithm one has to solve linear systems with many right–hand sides, we got important performance advantages through the somewhat unusual explicit computation of the matrix–product $P_{Jac}^{-1}A$. We have tested different iterative solvers, but they appear with respect to the overall performance as nearly as equivalent. Direct parallel solvers for banded systems can in no means compete with their iterative counterparts. The approach to reuse Krylov subspaces constructed in preceeding extrapolation steps appears as not very encouraging, at least for the examples we selected.

Our parallel version of LIMEX exhibits scalability up to 64 processors even for the medium–sized test problems. We have also demonstrated that the code is applicable in a large application package opening new possibilities for the simulation of chemical equipments. Thus our approach has resulted in highly scalable parallel programs for solving large systems of stiff ODEs and DAEs, which occur in many challenging applications.

The satisfying parallel efficiency we obtained for the LIMEX code motivate further research, some topics shoud be mentioned. At first we will investigate an embedding of our algorithms in the both in time and space adaptive extrapolation code PDEX1M by Nowak [34, 35]. Secondly we will examine generalizations of our approaches to higher dimensional problems. For this attempt we will most likely to change from exact subdomain solves to inexact ones. This implies the disadvantage that we can not use the reduced system technique and the advantage that we can profit from the simplified iterative algorithmic scheme explained in §2. Thirdly we have to adapt the stepsize and order order control mechanism to take the changed computational weights more into consideration.

References

[1] P. Arbenz, W. Gander: *A Survey of Direct Parallel Algorithms for Banded Linear Systems.* Technical Report TR 221, Institute for Scientific Computing, ETH Zürich (1994).

[2] Z. Bai, D. Hu, L. Reichel: *A Newton basis GMRES implementation.* IMA J. Numer. Anal. **14**, pp. 563–581 (1994).

[3] R. Barriuso, A. Knie: *SHMEM User's Guide for FORTRAN, Rev. 2.2.* Cray Research Inc. (1994).

[4] R.H. Bisseling: *Parallel iterative solution of sparse linear systems on a transputer network.* In: A.E. Fincham, B. Ford (eds.): Parallel Computation. Oxford University Press: Oxford (1993)

[5] P.E. Bjørstad: *Multiplicative and Additive Schwarz Methods: Convergence in the 2 domain case.* In: T. Chan, R. Glowinski, J. Périaux, O. Widlund (eds.): Domain Decomposition Methods. SIAM: Philadelphia, PA (1989).

[6] P.E. Bjørstad, O.B. Widlund: *To overlap or to not overlap: A note on domain decomposition method for elliptic problems.* SIAM J. Sci. Stat. Comp. **10**, pp. 1053–1061 (1989).

[7] F.A. Bornemann: *Adaptive multilevel discretization in time and space for parabolic partial differential equations.* Technical Report TR 89-7, Konrad-Zuse-Zentrum für Informationstechnik Berlin (1989).

[8] E. Brakkee: *A domain decomposition method for the advection-diffusion method.* Report 94-08, Faculty of Technical Mathematics and Informatics, Delft University of Technology, Delft (1991).

[9] E. Brakkee, K. Vuik, P. Wesseling: *Domain decomposition for the incompressible Navier–Stokes equations: solving subdomain problems accurately and inaccurately.* Report 95-37, Faculty of Technical Mathematics and Informatics, Delft University of Technology, Delft (1995).

[10] R. Bramley, V. Meñkov: *Low Rank Off–Diagonal Block Preconditioners for Solving Sparse Linear Systems on Parallel Computers.* Tech. Rep. 446, Department of Computer Science, Indiana University, Bloomington (1996).

[11] K. Burrage: *Parallel and Sequential Methods for Ordinary Differential Equations.* Oxford University Press: New York (1996).

[12] X.–C. Cai, W.D. Gropp, D.E. Keyes: *A comparison of some domain decomposition and ILU preconditioned iterative methods for nonsymmetric elliptic problems.* Numer. Lin. Alg. Appl. **1** (1994).

[13] D. Calvetti, J. Petersen, L. Reichel: *A parallel implementation of the GMRES method.* In: L. Reichel, R.S. Varga (eds.): Numerical Linear Algebra. de Gruyter: Berlin (1993).

[14] J. Cullum, A. Greenbaum: *Relations between Galerkin and Norm-Minimizing Iterative Methods for Solving Linear Systems.* SIAM J. Matrix Anal. Appl. **17**, pp. 223–247 (1996).

[15] R.D. da Cunha, T. Hopkins: A parallel implementation of the restarted GMRES iterative algorithm for nonsymmetric systems of linear equations. Adv. Comp. Math. **2**, pp. 261–277 (1994).

[16] P. Deuflhard: *Recent Results in Extrapolation Methods for ODEs.* SIAM Review, **27**, pp. 55–535 (1985).

[17] P. Deuflhard, U. Nowak: *Extrapolation Integrators for Quasilinear Implicit ODEs.* In: P. Deuflhard, B. Enquist (eds.): Large Scale Scientific Computing. Progress in Scientific Computing **7**, pp. 37–50. Birkhäuser (1987)

[18] P. Deuflhard, J. Lang, U. Nowak: *Adaptive Algorithms in Dynamical Process Simulation.* 8th Conference of the European Consortium for Mathematics in Industry, Kaiserslautern 1994.

[19] P. Deuflhard: *Order and Stepsize Control in Extrapolation Methods.* Numer. Math. **41**, pp. 399-422 (1983).

[20] P. Deuflhard: *Uniqueness theorems for siff ODE initial value problems.* In: D.F. Griffiths, G.A. Watson (eds.): Numerical Analysis 1989, Proceedings of the 13th Dundee Conference Juny 1989. Wiley: New York (1989).

[21] J.J. Dongarra, R.C. Whaley: *A Users' Guide to BLACS v1.0.* LAPACK Working Note DRAFT, ORNL (1995).

[22] J.J. Dongarra, A.H. Sameh: *On some parallel banded system solvers.* Parallel Computing **1**, pp. 223–235 (1984).

[23] P.F. Dubois, A. Greenbaum, G.H. Rodrigue: *Approximating the inverse of a matrix for use in iterative algorithms on vector processors.* Computing **22**, pp. 257–268 (1979).

[24] G. Eigenberger, U. Nieken: *Katalytische Abluftreinigung: Verfahrenstechnische Aufgaben und neue Lösungen.* Chem. Ing. Techn. **63(8)**, (1991)

[25] T.L. Freeman, C. Phillips: *Parallel Numerical Algorithms.* Prentice Hall: New York, London, Toronto, Sydney, Tokyo, Singapure (1992).

[26] G.H. Golub, C.F. van Loan: *Matrix Computations.* The John Hopkins University Press: Baltimore and London (1990).

[27] W.D. Gropp, B.F. Smith: *Experineces with domain decomposition in three dimensions: Overlapping Schwarz methods.* Technical report, Mathematics and Computer Science Division, Argonne National Laboratory, 1992.

[28] W.D. Joubert, G.F. Carey: *Parallelizable restarted iterative methods for nonsymmetric linear systems.* Int. J. Comput. Math. **44**, pp. 243–290 (1992).

[29] S.K. Kim, A. Chronopoulos: *An efficient Arnoldi method implemented on parallel computers.* 1991 International Conference on Parallel Processing, vol. III (1991).

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

[31] J. Lang, A. Walter: *A Finite Element Method Adaptive in Space and Time for Nonlinear Reaction–Diffusion Systems.* IMPACT Comput. Sci. Engrg. **4**, pp. 269-314 (1992).

[32] G.–C. Lo, Y. Saad: *Iterative Solution of General Sparse Systems on Clusters of Workstations.* Technical Report UMSI 96-117, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis (1996).

[33] Ch. Lubich: *Linearly implicit Extrapolation Methods for Differential–Algebraic Systems.* Numer. Math. **55**, pp. 129–145 (1989).

[34] U. Nowak: *Adaptive Linienmethoden für nichtlineare parabolische Systeme in einer Raumdimension.* Technical Report TR 93-14, Konrad-Zuse-Zentrum für Informationstechnik Berlin (1993).

[35] U. Nowak: *A fully Adaptive MOL–Treatment of Parabolic 1D–Problems with Extrapolation Techniques.* Appl. Num. Math. **20**, pp. 129–145 (1996).

[36] V. Pan, R. Schreiber: *An improved Newton iteration for the generalized inverse of a matrix, with applications.* SIAM J. Sci. Stat. Comput. **12**, pp. 1109–1130 (1991).

[37] N. Peters, J. Warnatz (eds.): *Numerical Methods in Laminar Flame Propagation.* Notes on Numerical Fluid Dynamics **6**, Vieweg (1982).

[38] G. Radicati di Brozolo, Y. Robert: *Parallel conjugate gradient–like algorithms for solving sparse nonsymmetric linear systems on a vector multiprocessor.* Parallel Computing **11**, pp. 223–239 (1989).

[39] T. Rauber, G. Rünger: *Load Balancing for Extrapolation Methods on Distributed Memory Multiprocessors.* In: Lecture Notes in Computer Science **817**, pp. 277-288, Athen, July 1994. PARLE 1994, Springer.

[40] Y. Saad, M.H. Schultz: *GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems.* SIAM J. Sci. Stat. Comp. **7**, pp. 856–869 (1986).

[41] Y. Saad: *Iterative Methods for Sparse Linear Systems.* PWS Publishing Co.: Boston (1996).

[42] K. Schaber: *Aerosol Formation in Absorption Processes.* Chem. Engng. Sci. **50**, pp. 1347–1360 (1995).

[43] K. Schaber, J. Körber: *Formation of Aerosols in Absorption Processes for Exhaust Gas Purification.* J. Aerosol. Sci. **22**, pp. S501–S504 (1991).

[44] J.N. Shadid, R.S. Tuminaro: *Sparse iterative algorithm software for large–scale MIMD machine: an initial discussion and implementation.* Tech. Rep., DOE's Massively Parallel Computing Research Laboratory, Sandia National Laboratories, Albuquerque (1991).

[45] E. de Sturler: *Incomplete Block LU Preconditioners on Slightly Overlapping Subdomains for a Massively Parallel Computer.* App. Num. Math. (IMACS) **19**, pp. 129–146 (1995).

[46] E. de Sturler, H. van der Vorst: *Communication cost reduction for Krylov methods on parallel computers.* In: Y. Saad (ed.): Numerical Methods for Large Eigenvalue Problems. Manchester University Press: Manchester (1992).

[47] V. Simoncini, E. Gallopoulos: *A hybrid block GMRES method for nonsymmetric systems with multiple right–hand sides.* J. Comput. Appl. Math. **66**, pp. 457–469 (1996).

[48] H.A. van der Vorst: *BI-CGSTAB: A fast and smoothly converging variant of BI-CG for the solution of non–symmetric linear systems.*
SIAM J. Sci. Stat. Comp. **13**, pp. 631-644 (1992).

[49] X. Xu, N. Qin, B. Richards: *Alpha–gmres – a new parallelizable iterative solver for large sparse nonsymmetrical linear systems arising from CFD.* Int. J. Numer. Meth. Fluids **15**, pp. 613–623 (1992).