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

Rainald Ehrig
Ulrich Nowak
Peter Deuflhard

Massively Parallel Linearly-Implicit Extrapolation Algorithms as a Powerful Tool in Process Simulation

Preprint SC 97-43 (September 1997)

Massively Parallel Linearly-Implicit Extrapolation Algorithms as a Powerful Tool in Process Simulation

Rainald Ehrig   Ulrich Nowak   Peter Deuflhard

Abstract

We study the parallelization of linearly–implicit extrapolation codes for the solution of large scale PDE systems and differential algebraic equations on distributed memory machines. The main advantage of these algorithms is that they enable adapativity both in time and space. Additive Krylov–Schwarz methods yield high parallel perfomance for such extrapolation methods. Our approach combines a slightly overlapping domain decomposition together with a polynomial block Neumann preconditioner and a reduced system technique. Furthermore we get important advantages through the explicit computation of the matrix–products of the preconditioner and the matrix of the linear system. The parallel algorithms exhibit scalability up to 64 processors already for medium–sized test problems. We show that the codes are really efficient in large application systems for chemical engineering problems.

Contents

1 Introduction

Many challenging applications in the engineering sciences lead to large systems of PDEs, which have to be solved together with complex constraints, like balance conditions or transfer equations as examples. Such differential algebraic equations (DAEs), which can be understood as differential equations on manifolds, have an increasing importance in the field of mechanical and chemical engineering, besides many others. The DAEs arising in such applications can mostly be written as

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

with an in general singular matrix–pencil $B(t,y)$. We will concentrate here on DAEs resulting from a space discretization of PDEs by the method of lines in the 1D-case.

Since most real world problems possess multiple time scales, we have to use implicit methods. A suitable algorithmic framework is the linearly–implicit Euler discretization

$$y_{k+1} = y_k + (B(y_k) - hA)^{-1} h f(y_k) \tag{2}$$

with $A = (f(y) - By')_y$, the Jacobian of the residual of (1). Combined with extrapolation this one–step method permits an adaptive stepsize and order control and is well established in a wide area of applications (codes LIMEX and PDEX1M), see DEUFLHARD [1], DEUFLHARD, LANG AND NOWAK [2] and NOWAK [3]. Within the extrapolation one computes for a basic stepsize H approximations $T_{j,1}$ for $y(t_0+H)$ using the described discretization with stepsizes $h_j = H/n_j, j = 1,\ldots,j_{max}$. Then the extrapolation tableau recursively defines higher order approximations $T_{j,k}$. As usual the subdiagonal differences $\varepsilon_j = \|T_{j,j} - T_{j,j-1}\|$ are taken as error estimates.

The efficiency of any serial or parallel implementation of the algorithms almost solely depends on the performance of two computational kernels: the evaluation of the Jacobian and particularly of the solution of the linear systems. In the following we describe the general concepts of a parallel extrapolation algorithm, targeted to a distributed memory architecture. Then we describe the steps leading to a highly scalable parallel solver and demonstrate the efficiency of our methodology.

2 General concepts

At first sight, a simple and promising method is to compute the approximations $T_{j,1}$ independently on different processors, but this approach would restrict the number of processors to the maximal order attainable by the extrapolation method. Furthermore we could not use the advanced order and stepsize control techniques, as in this procedure the order j_{max} is determined within the current step. Nevertheless this parallelization strategy is studied by some authors (BURRAGE [4], RAUBER [5]).

A more general applicable and commonly used strategy is the domain decomposition method, which does not involve any limitation to the number of processors. 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 obtained from solutions on suitably selected subdomains. Within the linearly–implicit extrapolation methods then it is required that the function f in the right–hand side of the DAE can be evaluated locally. This assumption is fulfilled for most problems resulting from a discretization of PDEs by the method of lines. If we could solve efficiently and in parallel the linear systems arising in the extrapolation, the whole extrapolation scheme can straightforward be parallelized, assigning the processors to different parts 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 in the computation of the error estimates.

3 A step by step development of a parallel solver

Given the distributed evaluation and storage of the Jacobian we have to search for a solution methodology, which likewise uses this partitioning. Parallel direct methods, even if one uses specialized algorithms for banded systems referring to the domain decomposition, are not competitive for higher numbers of processors, due to the imbalance of computation and communication. In contrast to this the partitioned matrix and vector storage for iterative solvers enables a quite effective parallel matrix by vector multiplication, which needs only some "small" data exchanges of the right–hand side vector between neighbouring processors. So as usual the right choices of a parallel preconditioner and an iterative solver decide, whether the battle for low computational cost and scalable parallelism can be won.

As a test problem we use throughout this chapter a system of 11 PDEs, which models the startup phase of an automobile catalytic converter (Nowak [3] and Eigenberger and Nieken [7]), with strongly varying time–dependent chemical dynamics and stiffness. The speedups are always measured by comparison with our fastest sequential implementation. The communication bases on the Cray shared memory access routines. All solvers are implemented on a Cray T3D using complete LU factorizations on the subdomains. As iterative solver we used GMRES, but other iterative methods as BICGSTAB are likewise effective.

3.1 Block Jacobi and block Neumann preconditioner

Standard sequential preconditioners such as ILU or SSOR are in most cases not appropriate in the parallel world. Their major bottleneck are the backsolves involving triangular distributed matrices. An alternative technique targeted more specifically at parallel environments is the block Jacobi preconditioner, or so–called additive Schwarz procedure. This very simple but highly scalable approach uses complete or incomplete factorizations on the subdomains assigned to the processors, which can be computed and applied completely in parallel.

An 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. As an example one gets for a truncated Neumann–series expansion

$$A^{-1} \approx (2I - P_{Jac}^{-1}A)\, P_{Jac}^{-1} =: P_{Neu} \,. \tag{3}$$

This preconditioner was introduced by Dubois et al. [8] and later studied by da Cunha et al. [9]. Obviously P_{Neu} requires one more matrix by vector multiplication and one additional application of the block Jacobi preconditioner.

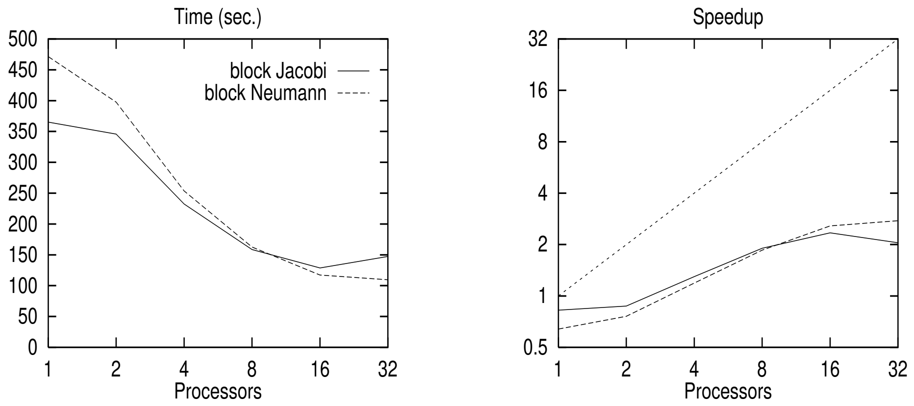

Figure 1: Performance results for the block Jacobi and Neumann preconditioner

The results of applying both the block Jacobi or Neumann preconditioner are at first glance not encouraging, see Fig. 1. One obtains only a speedup of about 2.3 with 16 processors, the number of iterations needed by GMRES increases rapidly with the number of processors. For the block Jacobi the solution exhibits spurious oscillations around the processor boundaries, which indicates that the coupling across the boundaries is not enough considered, whereas the block Neumann method shows a satifying robustness.

The block preconditioners can also be defined through other mathematical approaches. First the classical Richardson matrix–iteration produces as iterates exactly the values of the truncated Neumann series. Thus the (block) Jacobi preconditioner is the first, the (block) Neumann preconditioner is the second Richardson iterate. The application of these preconditioners within an iterative method can therefore be seen as a nested iterative scheme, the inner iterations are the 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 vector–valued equivalent of the function $f(x) = 1/x - a$ (Pan and Schreiber [10]). The different interpretations suggest a variety of modifications and strategies especially for higher–dimensional cases.

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

One common property of the block Jacobi and Neumann approach used in iterative solvers 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 in Fig. 2. Herein

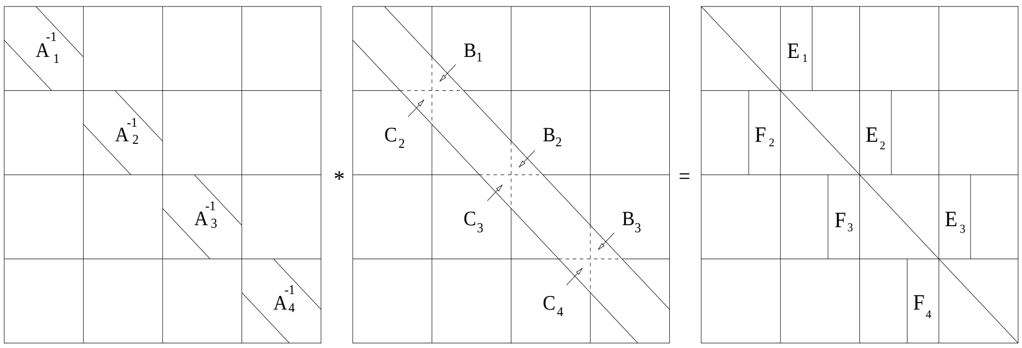

Figure 2: Schematic representation of the matrix product $P_{Jac}^{-1}A$ on 4 processors

the E_i and F_i are defined by $E_i = A_i^{-1}B_i$ and $F_i = A_i^{-1}C_i$, the diagonal elements of $P_{Jac}^{-1}A$ are equal to 1. Each of the matrices E_i and F_i covers m (or $(m+1)/2$, if A is block–tridiagonal) vectors of the length n/p, with n the size of the system and m the upper and lower bandwidth. The structure of $P_{Jac}^{-1}A$ resembles to the Schur complement technique. Indeed our algorithm can be understood as a generalized Schur complement method 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 that the local updates of the right–hand side vector are already done. So the computation of the matrices E_i and F_i is very efficient, as illustrated by the performance results in Fig. 3.

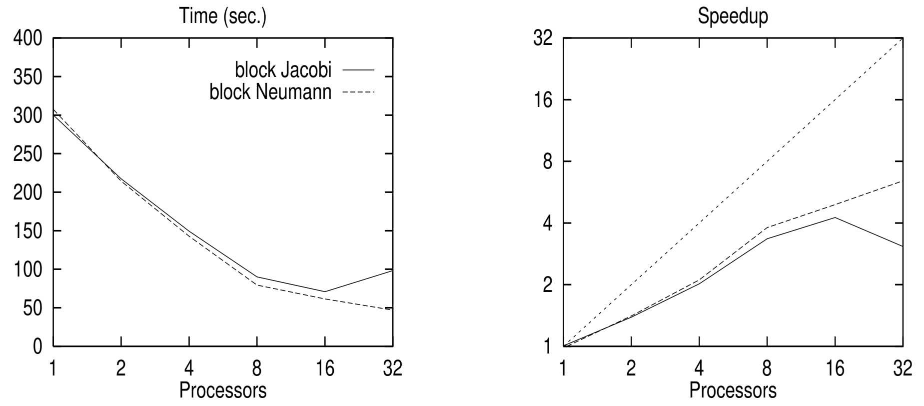

Figure 3: Performance results for the block Jacobi and Neumann preconditioner using explicit computation of $P_{Jac}^{-1} A$

3.3 Overlapping domain decomposition

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 domain decomposition. 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

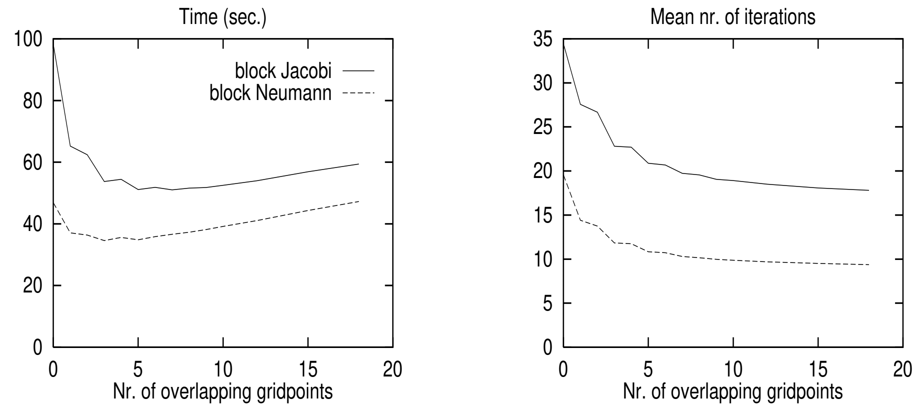

Figure 4: Performance results as a function of the size of the overlap region for the block Jacobi and Neumann preconditioner with averaging on 32 processors

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. The advantage of averaging is that the vectors used to iterate are consistent on all processors. The algorithmic realization of these operators needs no additional calls of communication routines, only the amount of data exchanged between neighbouring processors increases depending on the size of the interface region.

Applying the overlapping domain decomposition gives a remarkable rise of efficiency especially for the block Jacobi preconditioner, see Fig. 4. With a definition of 5 gridpoints as overlap region the computing times and the 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. 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%.

3.4 The 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 be the submatrices introduced in 3.2. 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 equivalently the vectors x and b, see Fig. 5.

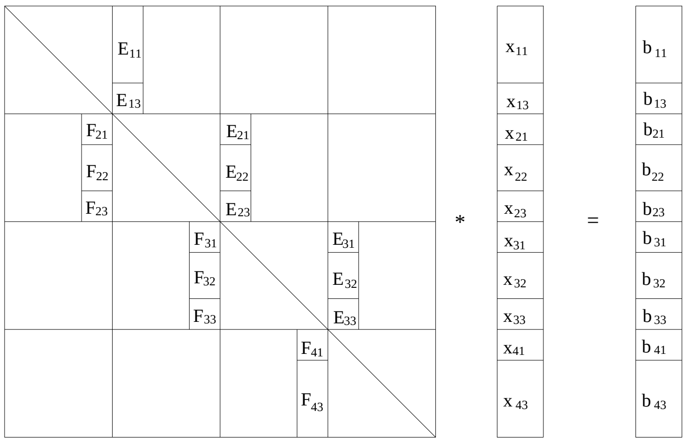

Figure 5: Partitioning of the linear system $P_{Jac}^{-1}Ax = b$ on 4 processors

Now we can split up the whole linear system in a set of equations with the unknowns $x_{13}, x_{21}, x_{23}, \ldots, x_{p1}$ operating only of the processor boundaries. These equations build the so–called *reduced system* of order $2m(p-1)$ and is much smaller than the original system. It can be solved completely independent of the set of remaining equations with the unknowns $x_{11}, x_{22}, x_{32}, \ldots, x_{p-1,2}, x_{p3}$. This system 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. The convergence properties of an iterative method applied to the full matrix–product or the reduced system are very similar. The eigenvalues of the reduced system are eigenvalues of $P_{Jac}^{-1}A$ as well, and this matrix has the additional eigenvalue 1.

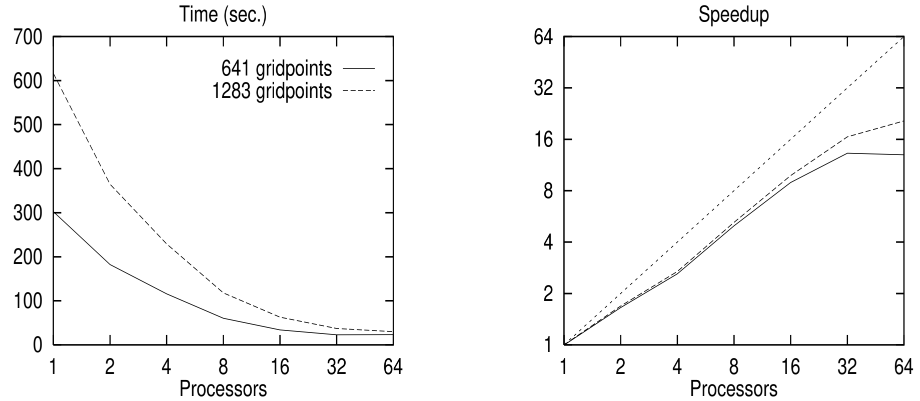

Figure 6: Performance results with the reduced system technique on two gridsizes

Again there are some related mathematical techniques which we should mention. The inherent equivalence between these aproaches is not commonly recognized. First the reduced system method is introduced in detail in [11] to develop parallel direct solution methods for banded systems. Furthermore Brakkee et al. [12, 13] investigated in the context of the 2D Navier–Stokes equation domain decomposition methods and derived a so–called interface–equation in connection with exact subdomain solutions, which is completely equivalent to the reduced system. In a more general manner, Bramley and Meñkov [14] have examined low rank off–diagonal block preconditioners. They studied approximations for a sparse matrix which can be written as $B = D + UV^T$, U and V matrices composed of only a "few" vectors. Our approch can be embedded in their terminology, since one easily constructs "small" matrices U and V with $P_{Jac}^{-1}A = I + UV^T$. Then $I + V^TU$ is exactly the reduced system.

Applying the described reduced system technique combined with an overlapping domain decomposition we once more obtain a substantial gain of performance, see Fig. 6. Because the iterations now are very cheap, we get the minimal computing time, if the overlap region consists of a single grid point. Fig. 6 demonstrates now clearly the scalability of our step–by–step evolved algorithm already for the only medium–sized testproblem.

4 Numerical Results

In this section we demonstrate the performance and scalability of our approaches when applied to large and difficult problems from chemical engineering.

4.1 Results for an aerosol formation problem

The purpose of the program SENECA (Simulation with Extrapolation methods and NEwton techniques of Chemical and technical processes with Aerosol formation), a joint project with the University Karlsruhe, is a general tool to simulate stationary and instationary processes in chemical equipments including aerosol formation. The 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 [15, 16] for a detailed description.

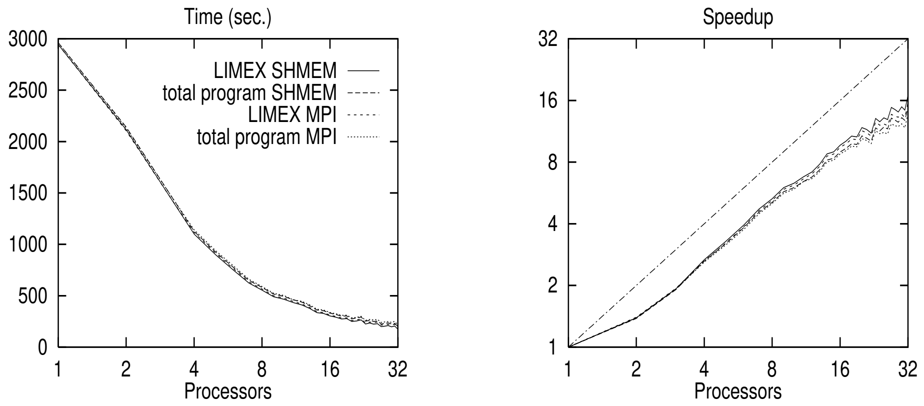

Figure 7: Performance results for the aerosol formation problem

The 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 nonlinear equations for heat and mass transfer as well as for the volume, mass and energy balances. The space discretization of the PDEs yields a set of DAEs, 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 also the size of the linear systems changes after each integration step. Due to the spatially 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 block tetra–diagonal linear systems. Regardless of these difficulties, all algorithmic strategies we have developed can be applied without any principial changes. Within the program package, the LIMEX code is only one of many subroutines, even if by far the most time consuming one. We have made no attempts to parallelize the whole package. In order to be consistent, we therefore have to distribute the solution vector over all processors after each integration step.

We present in Fig. 7 the performance results on a Cray T3E for a medium problem size. Hereby the mean gridsize, averaged over the whole integration is about 150, the size of the linear systems is about 8000. The block sizes fluctuate between 10 and 46. We have included the results for a MPI–based implemetation, but the differences are not very significant.

The results clearly show the value of the parallel extrapolation method within application specific codes for the integration of large systems of DAEs. 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.

4.2 Results for the adaptive program PDEX1M

We demonstrate finally, that the algorithmic approaches are even successfully applicable in the adaptive solver PDEX1M for parabolic differential equations (Nowak [3]). This algorithm is fully adaptive in space and time. The number of grid points required for a certain solution accuracy as well as the distribution of grid points and the time steps are automatically adjusted, based on the relative errors for the time and space discretization. These errors are estimated at each time step via extrapolation

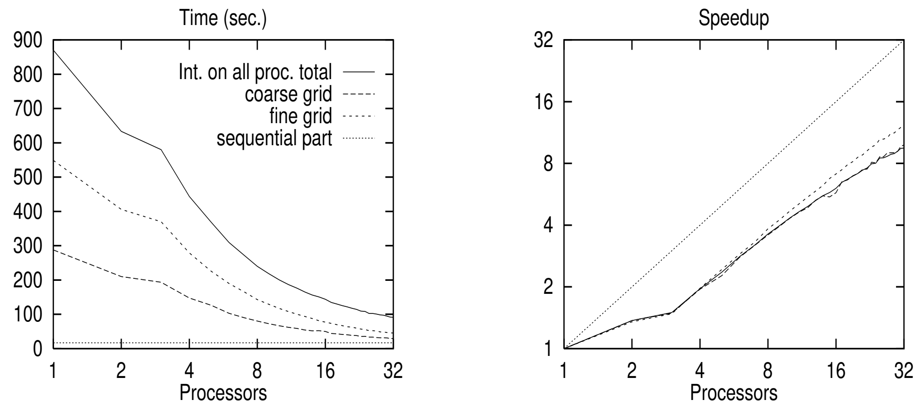

Figure 8: Performance results of PDEX1M for the catalytic ignition

and used for a local regridding procedure. Based on the local errors estimates global error norms are computed which enables a separate error control in space and time. The program incorporates advanced techniques like moving grid options.

We present in Fig. 8 preliminary results (the interpolation between the different grids is not yet fully parallelized) of an example from the University Stuttgart, where based on this code the application package PDEXPACK has been developed, Nowak et al. [6]. It simulates the catalytic ignition during methane oxidation on platinum.

To get a high parallel performance our implementation allows to integrate on the coarse resp. fine grid using different processor sets in the relation 2:1. Then one can obtain a better scalability even for medium grid sizes. This expectation is indeed partially fulfilled for our test problem, see Fig. 9.

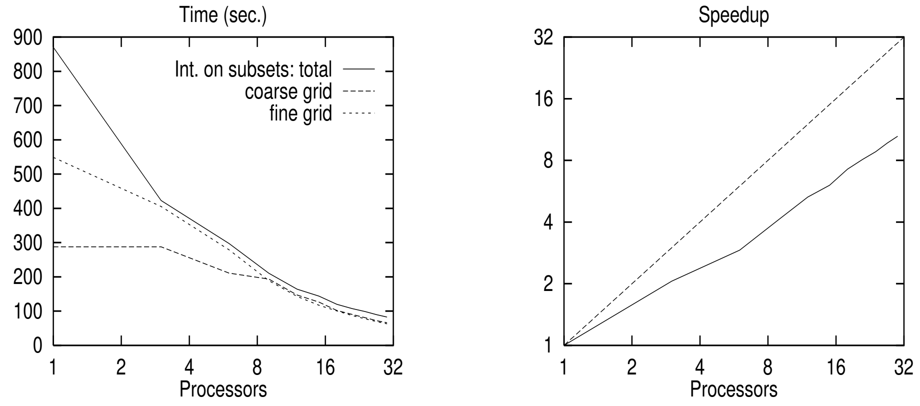

Figure 9: Performance results of PDEX1M using different processor sets for coarse and fine grid integration

5 Parallel implementation issues

We implemented our algorithms on the Cray T3D with 256 processors (64 MB memory) and the Cray T3E with 128 processors (128 MB) at the Konrad–Zuse–Zentrum. 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. The order and stepsize control are done by all processors simultaneously.

All message passing calls are implemented through the shared memory access communication routines (SHMEM library) from Cray Research Inc. 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 routine `shmem_put`, which is the fastest point–to–point communication routine on the T3D. Because the codes itself never calls directly any communication function the programs can be switched easily to any other message passing library, especially to MPI.

5.1 An heuristic scalability analysis

We conclude this section with an analysis of the scalability of our algorithms, but not by counting Mflops or computing serial or parallel complexities. This approach is of very limited value for a whole application code, since it does not refer to the efficiency of assembly coded linear algebra kernels to the communication performance of a specific parallel machine. Instead we have measured the computing times of the main parts of our program to demonstrate the different scalability potential, see Fig. 10. The application dependent parts scale very well, i.e. the calls of the right hand side f within the extrapolation and the evaluation of the Jacobian, likewise even the

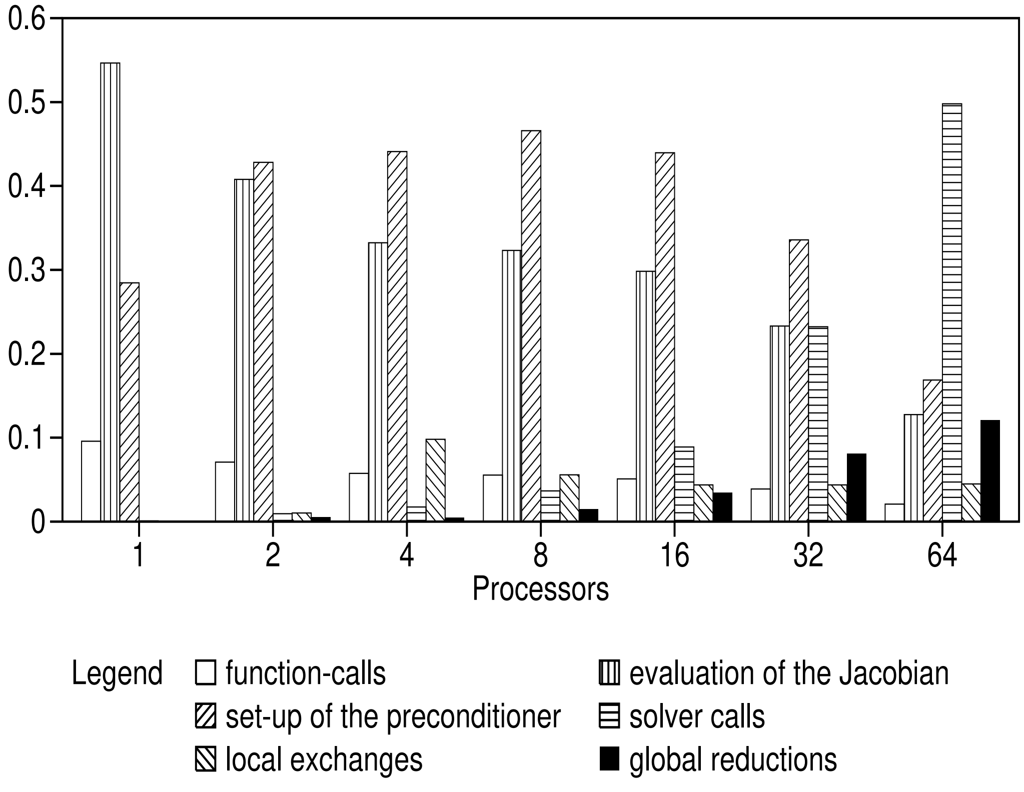

Figure 10: Relative contributions to the computing time

computation of the preconditioner, which can be done almost completely in parallel. An ideal scalability is prevented by the communication calls and in particular by the iterative solver due to the increasing number of iterations. Indeed with 64 processors the solver calls predominate the computing time, whereas the application specific parts become more and more insignificant. This processor number, however, is not a general limit for the scalability of our methods, as for the medium–sized test problem we have selected. For larger examples we 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/T3E are the ideal target machines for parallel extrapolation codes. This is proven evidently by the fact, that within the GMRES algorithm we get no considerable differences between the so–called modified Gram–Schmidt orthogonalization, which is numerical more stable, but needs much more global reductions and the classical counterpart, where all communication can be done at once. As an counterexample Lo and Saad [17] reported on clusters of workstations differences of sometimes more than 50% between the overall performance of the classical resp. modified procedure.

6 Conclusions

We have demonstrated that the parallel extrapolation algorithms open new possibilities for realistic simulations in chemical engineering and other areas as well. We finally note, that the development of highly scalable and efficient parallel algorithms remains a mathematical chal lenge, regardless of the existence of parallel toolkits or high–speed communication libraries.

Our efforts for the development of a highly scalable and efficent solver for large systems of DAEs are not finished. Some of the open questions will briefly be mentioned.

For the GMRES–algorithm we should consider, that one solves within the extrapolation scheme *many* linear systems, some of them with the same matrices, some of them with matrices different mainly in the diagonal. This situation suggests to reuse the work done in the preceeding steps, e.g. the already constructed Krylov spaces. Until now our preliminary approaches did not result in a remarkable gain of performance, but such methods should get our interest again for higher dimensional problems, where direct subdomain solutions may be too expensive.

Furthermore we have to consider, that the actual interesting property of the approximate solutions of the linear systems is the error, not the (preconditioned) residual. So the development of error minimizing iterative methods would be of great value.

In many real applications for DAEs, a serious problem is the determination of consistent initial values. To do this likewise in parallel, we have developed a method of extrapolated Newton–iterations, which can be imbedded in the LIMEX–code and which is highly efficient in the SENECA package.

All these considerations will influence even the evolution of the *sequential* algorithmic counterparts. So the efforts made for a parallel code are always an opportunity to reflect upon the algorithmic foundations of a given numerical method.

References

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

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

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

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

[5] 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.

[6] U. Nowak, J. Frauhammer, U. Nieken: *A fully adaptive algorithm for parabolic partial differential equations in one space dimension.* Comp. Chem. Eng. **20**, pp. 547–561 (1996).

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

[8] 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).

[9] 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).

[10] 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).

[11] 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).

[12] 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).

[13] 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).

[14] 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).

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

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

[17] *Iterative Solution of General Sparse Systems on Clusters of Workstations.* Technical Report UMSI 96-117, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis (1996).