Konrad-Zuse-Zentrum
für Informationstechnik Berlin

Takustraße 7
D-14195 Berlin-Dahlem
Germany

ULRICH NOWAK, RAINALD EHRIG, LARS OEVERDIECK

Parallel Extrapolation Methods and Their Application in Chemical Engineering

Preprint SC 97-69 (December 1997)

Parallel Extrapolation Methods and Their Application in Chemical Engineering

Ulrich Nowak, Rainald Ehrig, Lars Oeverdieck

10th February 1998

Abstract

We study the parallelization of linearly–implicit extrapolation methods for the solution of large scale systems of differential algebraic equations arising in a method of lines (MOL) treatment of partial differential equations. In our approach we combine a slightly overlapping domain decomposition together with a polynomial block Neumann preconditioner. Through the explicit computation of the matrix products of the preconditioner and the system matrix a significant gain in overall efficiency is achieved for medium–sized problems. The parallel algorithm exhibits a good scalability up to 32 processors on a Cray T3E. Preliminary results for computations on a workstation cluster are reported.

1 Introduction

Many challenging applications in the engineering sciences lead to systems of time–dependent partial differential equations (PDEs), which have to be solved together with complex constraints, like balance conditions or transfer equations as examples. In this paper we consider large systems of n PDEs in one space dimension $x \in [x_L, x_R]$, which can be written in the form

$$B(x,t,u,u_x)u_t = f(x,t,u,u_x,(D(x,t,u)u_x)_x)\ , \tag{1}$$

where the matrix B may contain zero rows. A popular technique for the numerical solution of (1) is the method of lines. In this approach the space derivative terms are replaced by appropriate discrete approximations, e.g. finite differences. With that a large scale system of ordinary differential equations (ODEs) or differential algebraic equations (DAEs) is generated. In most cases it is of the form

$$\mathbf{B}(t,\mathbf{y})\mathbf{y}' = \mathbf{f}(t,\mathbf{y}), \tag{2}$$

where the matrix $\mathbf{B}$ may be singular. This system is nonlinear, stiff and block structured. Its numerical solution can, in principle, be attacked by any DAE integrator. Among the most efficient integrators of this type are the implicit multistep code DASSL [10], the implicit Runge-Kutta code RADAU5 [8], and the linearly implicit extrapolation code LIMEX [4]. We prefer the latter code, since it requires – because of its linearly implicit structure – one iteration loop less than implicit codes. A further structural advantage is the one–step nature of this method, which allows changes in the the problem formulation after each integration step.

2 Time Discretization

The time discretization of LIMEX is based on the elementary linearly implicit Euler discretization

$$(\mathbf{B}(t,\mathbf{y}_k) - h\mathbf{J})(\mathbf{y}_{k+1} - \mathbf{y}_k) = h\mathbf{f}(t_{k+1},\mathbf{y}_k) \ , \tag{3}$$

where $\mathbf{J} \approx \frac{\partial}{\partial \mathbf{y}}(\mathbf{f} - \mathbf{B}\mathbf{y}')\,|_{t=t_0}$ is the (approximate) Jacobian of the residual of (2). Typically, this matrix shows a block band structure. Combined with extrapolation this one–step method permits an adaptive stepsize and order control and is well established in a wide area of applications, see e.g. Deuflhard, Lang and Nowak [5] and Nowak [9]. Within the extrapolation process 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 $T_{j,k}$

$$T_{j,k} = T_{j,k-1} + \frac{T_{j,k-1} - T_{j-1,k-1}}{n_j/n_{j-k+1} - 1} \ , \quad k = 1,\ldots,j \ . \tag{4}$$

As usual the subdiagonal differences $\varepsilon_j = \|T_{j,j} - T_{j,j-1}\|$ are taken as error estimates. One basic integration step may be sketched as follows

Compute Jacobian $J = (f - By')_y$
for $j = 1,\ldots,j_{max}$ *while convergence criterion not satisfied*
  $h_j = H/n_j$
  for $k = 0,\ldots,j-1$
    $$y_{k+1} = y_k + (B(t_{k+1},y_k) - h_j J)^{-1} h_j f(t_{k+1},y_k) \tag{5}$$
  $T_{j,1} = y_j$
  if $j > 1$ *compute* $T_{j,j}$ *and check convergence*
$y_{new} = T_{j,j}$

3 Parallelization Approaches

3.1 General Concepts

One may distinguish two different basic approaches for the parallelization of (5).

First, a simple and promising method is to compute the approximations $T_{j,1}$ independently on different processors. However, 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. Finally, a good load balancing is hard to achieve. For more information on this parallelization strategy we refer, e.g., to Burrage [2] and Rauber [12].

A more general applicable and commonly used strategy is the domain decomposition method, see e.g. [11] for details.. This approach refers directly to properties of the underlying physical problem. It is required that the function $\mathbf{f}$ (and the matrix $\mathbf{B}$) of our problem (2) can be evaluated locally. This assumption is fulfilled for most problems resulting from a discretization of PDEs by the method of lines.

As can be seen from (4) and (5), except for the linear system solution, the discretization and extrapolation process can be parallelized in a straightforward manner. The order and stepsize control algorithm requires the calculation of a few global norms only. Based on the local evaluation property the numerical difference approximation of the Jacobian can be parallelized easily and the matrix will be stored quite naturally in a block column format.

So, we have to search for a linear system solution methodology, which likewise uses this partitioning. Parallel direct methods, see e.g. ARBENZ AND GANDER [1], are in general not suited for our problem class. Even if one uses specialized algorithms for banded systems referring to the domain decomposition, they are not competitive for higher numbers of processors, due to the imbalance of computation and communication. In contrast to this, iterative solvers are suited very well for parallelization as the partitioned matrix and vector storage enables a quite effective parallel matrix by vector multiplication. Only some "small" data exchanges of the right–hand side vector between neighbouring processors are needed. However, the convergence properties of iterative solvers are highly determined by the choice of an appropriate preconditioner.

3.2 Preconditioning

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 triangularly 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

$$A^{-1} \approx (2I - P_{Jac}^{-1}A)\, P_{Jac}^{-1} =: P_{Neu} \ . \tag{6}$$

This preconditioner was introduced by DUBOIS ET AL. [6] and later studied by DA CUNHA ET AL. [3]. Obviously P_{Neu} requires one more matrix by vector multiplication and one additional application of the block Jacobi preconditioner.

The results of applying both the block Jacobi or Neumann preconditioner are at first glance not encouraging. Mainly, as the number of iterations needed increases rapidly with the number of processors. With the block Jacobi preconditioner the solution shows oscillations around the processor boundaries. This indicates that the coupling across the boundaries should appear in the preconditioner. The block Neumann method shows a satisfying robustness.

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. We use an averaging technique in order to guarantee consistency of the values in the overlap region. The algorithmic realization 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. With a definition of 5 gridpoints as overlap region the number of iterations, e.g. for GMRES [13], is reduced by approximately 50%. Even with just one overlapping grid point the performance is very satisfactory.

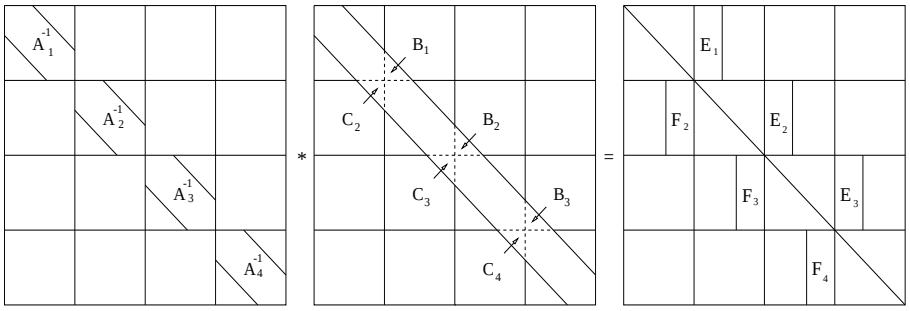

Figure 1: Schematic representation of the matrix product $P_{Jac}^{-1} A$ on 4 processors

3.4 Explicit Computation of $P_{Jac}^{-1} A$

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. 1. Herein 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. So, the application of the whole operator $P_{Jac}^{-1} A$ is reduced to some "small" matrix by vector multiplications. Both processes, the computation of E_i and F_i as well as the application of $P_{Jac}^{-1} A$ are fully parallelizable.

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.

3.5 The Reduced System Technique

Inspection of the E_i, F_i–representation of $P_{Jac}^{-1} A$ shows that the linear system to be solved can be split up in a set of equations with unknowns $x_{13}, x_{21}, x_{23}, \ldots, x_{p1}$ operating only on the processor boundaries, see Fig. 2. These equations build the so–called *reduced system* of order $2m(p-1)$ and this system is much smaller than the original system (m is the upper and lower bandwidth). 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 very efficiently. The reduced system itself is a distributed block tridiagonal matrix

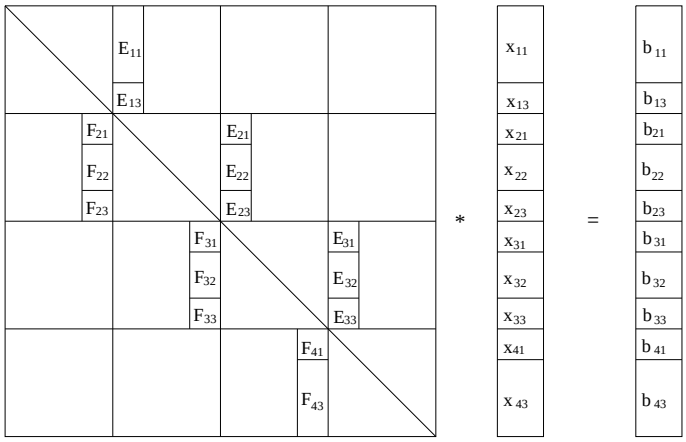

Figure 2: Partitioning of the linear system $P_{Jac}^{-1} A\, x = b$ on 4 processors

with blocks of order $2m$. This splitting technique works also for structural unsymmetric band matrices and in combination with an overlapping domain decomposition. To solve the reduced system we use a parallelized iterative method. We have experimented with various methods and achieved best results with GMRES and BICGSTAB [15].

3.6 Load Balancing

For a classical MOL treatment it is sufficient to distribute the computational work to the different processors at the very beginning of the computation only. If the processors are identical and can be used exclusively this means to subdivide the number of grid points, n_x, by the number of processors, p, and then to distribute the initial values to p subdomains each with approximately $\lfloor n_x/p \rfloor$ nodes x_i. This can be done easily by running from left to right through the nodes x_i of the overall computational domain. There will be as slight unbalancing as, in general, $n_x/p \notin \mathbb{N}$. However, as long as $n_x >> p$, this does not disturb the scalability of the parallel method.

In an adaptive MOL treatment one may dynamically adapt the number and location of the spatial discretization points in order to increase robustness and efficiency of the space discretization scheme. So, after each time integration step the assignment of grid points to processors must be rearranged. As the amount of data to be communicated is comparatively small and the load per grid point is still equidistributed the simple distribution scheme just mentioned works still very effective.

Things change if the dimension of the underlying PDE problem varies non-uniformly in the space coordinate x and, eventually, changes from time step to time step. Finally, if we aim to run our parallel method on heterogeneous workstation clusters we have to cope with the problems of processors with different computational power and non–exclusive access.

The main difficulty in the development of an appropriate dynamic load balancing procedure is the non–uniform problem dimension $n(x_i)$. Analyzing the computational kernels of our method it turns out that, counting operations, the

dimension n enters linearly and quadratically. The relative weights of the different parts depend on the hardware/software configuration of the target machine. As an example, the fact whether there are available machine optimized BLAS routines determines the relative contributions of the n-depending parts to the overall work count. To overcome these difficulties we use a simple a heuristical model for the (expected) normalized computational work per subdomain

$$W_q = \sum_{i=i_q^L}^{i_q^R} n_i^{\alpha_q} \quad , q = 1, ..., p \tag{7}$$

where i_q^L, i_q^R are the indices of the boundary grid points of the subdomain associated to processor q. The α_q are hardware/software depending constants. We found $\alpha_q = 0.2$ and $\alpha_q = 2.$ to be suitable values for a CRAY T3E and a SUN workstation cluster respectively. Obviously, $W_\Sigma = \sum_{q=1}^{p} W_q$ is the overall computational work to be done.

In order to deal with non–exclusive access to the processors we have to get the time dependent information on the computational power available for our process. So, we measure the real time Δt_q which is required by processor q to do the subdomain computation (without communication and waiting) for one time integration step. We now define its efficiency by setting

$$E_q = W_q / \Delta t_q \quad , q = 1, ..., p \quad . \tag{8}$$

The overall efficiency is given by $E_\Sigma = \sum_{q=1}^{p} E_q$. Our aim is to determine $i_q^L, i_q^R, q = 1, ..., q$ such that Δt_q is uniformly distributed over all processors. An optimal time for the current step would have been

$$\Delta t^{opt} = W_\Sigma / E_\Sigma \quad . \tag{9}$$

From this we get the optimal load for each processor by

$$W_q^{opt} = E_q \Delta t^{opt} = W_q \Delta t^{opt} / \Delta t_q. \tag{10}$$

With that our dynamic load balance procedure reads as follows. We start with a uniform (or better) initial domain decomposition. We do one time step measuring Δt_q. Then $W_q, W_\Sigma, E_q, E_\Sigma$ are computed and Δt^{opt} and W_q are determined by (9) and (10). In a final step a new domain decomposition is created by running from left to right through the computational grid, setting the new indices for the boundary nodes i_q^L, i_q^R by the requirement $W_q^{new} \approx W_q^{opt}$. This assignment procedure takes some constraints into account, e.g. we force that the memory requirement to processor q is less than the actual available memory and that there are enough grid points to allow a minimal domain overlap. This load adaptation procedure is used for all further time steps

4 Numerical Results

4.1 Catalytic converter problem

Our first application problem is a system of 11 parabolic PDEs. The equations model the startup phase of an automobile catalytic converter. A detailed description of the problem can found in [9]. This specific problem is a typical

example out of a challenging problem class, see EIGENBERGER AND NIEKEN [7]. After space discretization with centered second order finite differences on a uniform grid with 641 and 1283 gridpoints respectively, a system of 7051 (14113) DAEs has to be solved. In Fig. 3 the performance results are displayed for a CRAY T3E. Note that running on one processor only, the parallel version is, by construction, algorithmically equivalent to the sequential direct version.

The results demonstrate the good scalability of our parallel method already for this only medium–sized testproblem. Especially if the number of required grid points increases more than 32 processors could be used very efficiently.

4.2 Aerosol Formation Problem

In a joint project with the University of Karlsruhe, a software package (SENECA) for the simulation of processes in chemical equipments including aerosol formation is under development. The models include a detailed description of the formation and growth of fog droplets, see SCHABER AND KÖRBER [14]. This leads to a system of coupled hyperbolic and parabolic PDEs which has to be solved together with highly nonlinear equations for certain balance conditions. The space discretization of the PDEs yields a set of DAEs, where the Jacobian exhibits an unsymmetric block tetra diagonal structure.

The modeling 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. The number of necessary classes depends on the current state and varies strongly with the space coordinate x. Thus the number of PDEs *and* gridpoints may change from step to step. This can lead to severe load balancing problems, if the grid is not appropriately partitioned. For the computation on a CRAY T3E we use a simplified version of our dynamic load balancing procedure taking into account the likeness and exclusive use of the processors. Results for a system of moderate size are given in Fig. 4. Hereby the mean gridsize, averaged over the whole integration is about 150, the size of the linear system is about 8000. Included are the results using the Cray shared memory library SHEM and the results of a MPI-based implementation, which are very similar. As the LIMEX integration is the dominating part of the overall computation there is no significant difference comparing the results of the whole package with the

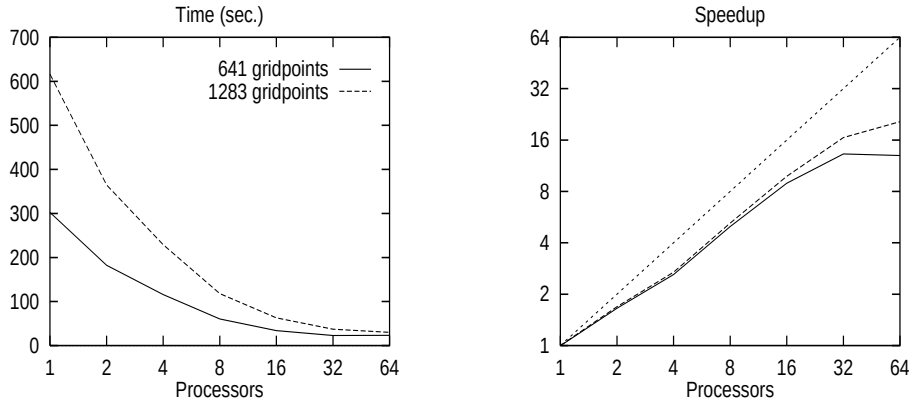

Figure 3: Performance results for the catalytic converter problem on two gridsizes

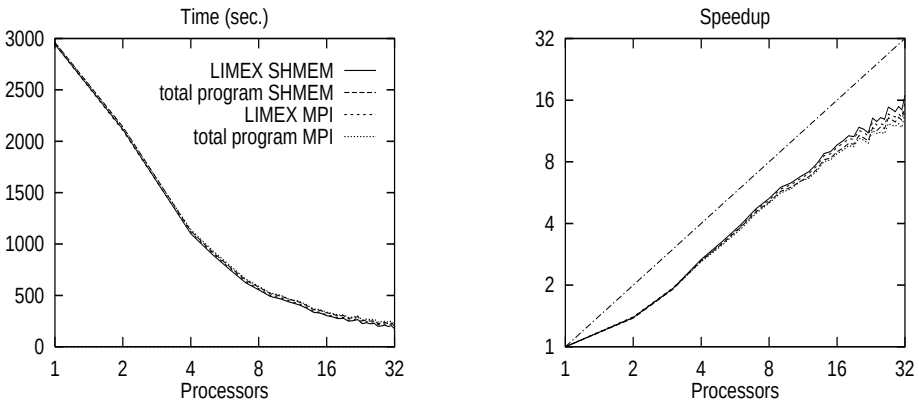

Figure 4: Performance results for the aerosol formation problem

performance results of the LIMEX part only. So, there is (currently) no need to parallelize the sequential parts of the package, e.g. the load balancing part or the model setup part.

Even if the number of processors is still limited by only medium sized grids the attainable speedup already enables the simulation of more complex problems.

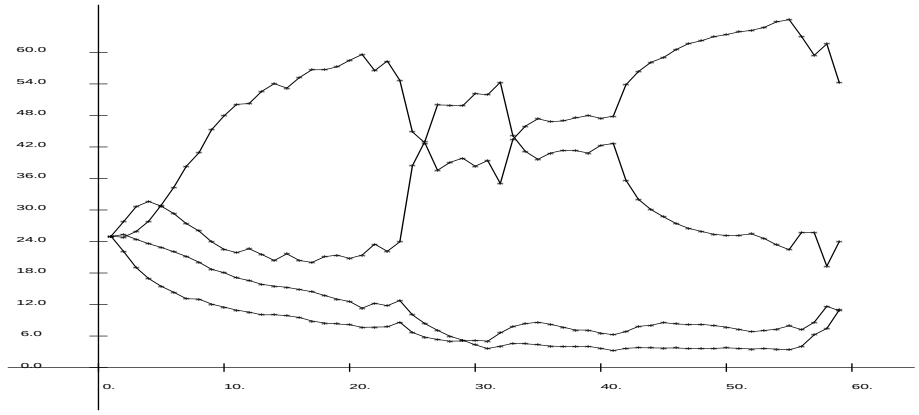

Figure 5: Dynamic load balancing on a heterogeneous workstation cluster

In order to study the aerosol formation in more detail the number of droplet classes must be increased. We expect several hundreds of PDEs per grid point. These problems have huge memory requirements and, as we need at least 4 grid points per subdomain, can not be run on the current CRAY T3E of the Konrad–Zuse–Center (ZIB). Alternatively, a powerful workstation cluster may be used. Currently we use a local network at ZIB which consists of 8 machines with different speed (4 – 35 MFLOP) and memory equipment (64MB – 1GB). Although our dynamic load balance procedure is quite simple, the first results are very satisfactory. Fig. 5 shows the load (in percent) on a heterogeneous cluster of 4 machines. Starting from a uniform distribution a nearly optimal

distribution evolves, i.e. the different relative speed of the machines is taken into account. To check the procedure further, after 21 time steps an external process is started on the most powerful processor. The load associated to this processor is automatically reduced and reset to the optimal value when the external process vanishes.

5 Conclusion

We have developed a parallel linearly–implicit extrapolation method. For tightly coupled distributed memory architectures the method shows a good scalability and overall efficiency for an interesting class of application problems. First steps towards an application on workstation clusters are encouraging. However there are still open questions which will be investigated in the near future.

References

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

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

[3] 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).

[4] P. Deuflhard, U. Nowak: *Extrapolation Integrators for Quasilinear Implicit ODE's.* In: P. Deuflhard, B. Engquist (eds.): Large Scale Scientific Computing. Progress in Scientific Computing, Birkhaeuser **7**, pp. 37–50 (1987).

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

[6] 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).

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

[8] E. Hairer, G. Wanner: *Solving Ordinary Differential Equations II. – Stiff and Differential–Algebraic Problems.* Springer Series in Computational Mathematics **14**, Springer Verlag, Berlin–Heidelberg (1991)

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

[10] L.R. Petzold: *A Description of DASSL: a differential–algebraic system solver.* Proc. IMACS World Congress, Montreal, Canada (1982).

[11] 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).

[12] 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.

[13] 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).

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

[15] 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).