Weierstraß–Institut
für Angewandte Analysis und Stochastik

im Forschungsverbund Berlin e.V.

Preprint ISSN 0946 – 8633

Parallelized numerical methods for large systems of differential–algebraic equations in industrial applications

Jürgen Borchardt, Friedrich Grund and Dietmar Horn

Preprint No. 382

Berlin 1997

Parallelized numerical methods for large systems of differential–algebraic equations in industrial applications

Jürgen Borchardt, Friedrich Grund and Dietmar Horn, Berlin

Summary. Based on a hierarchical modular modeling the large nonlinear systems of differential algebraic equations arising from industrial applications in electric circuit simulation or in dynamic process simulation of chemical plants can be structured into subsystems. Parallelized numerical methods for solving such systems are considered at the level of nonlinear and linear equations. Merging subsystems to blocks and extending the systems of nonlinear equations resulting from backward differentiation formulas block–structured Newton–type methods can be used for their solution on parallel computers. A parallelized Gaussian elimination method using pseudo code techniques for the LU–factorization of the large sparse systems of linear equations is implemented. The methods are successfully used on parallel and vector computers for the time domain simulation of VLSI circuits as well as for the dynamic process simulation of complex chemical production plants.

AMS Subject Classification: 65Y05, 65L05, 65H10, 65F50, 80A30, 92E20.

Keywords: Systems of differential–algebraic equations; Block partitioned systems; Newton–type methods; Gaussian elimination; Sparse–matrix techniques; Parallelization; Electric circuit simulation; Chemical process simulation.

1 Introduction

For the time domain simulation in many industrial applications structural properties of the considered objects are used for a hierarchical modular modeling. Thus electronic circuits usually consist of identical repetitive subnetworks as inverter chains, adders, or NAND– and NOR–gates. Analogously, complex chemical production plants are networks of coupled process units as pumps, reboilers, condensers, or trays of distillation columns. To each subcircuit or unit a mathematical model is assigned and they are linked to form the electric circuit or chemical plant respectively. For the time domain simulation this leads to initial value problems for large nonlinear systems of differential–algebraic equations (DAE's) which are structured corresponding to the subcircuits or units into m coupled subsystems

$$F_i(t, y(t), \dot{y}(t), u(t)) = 0, \;\; i = 1(1)m, \tag{1}$$

$$F_i : \mathbb{R} \times \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^q \to \mathbb{R}^{n_i}, \quad \sum_{i=1}^{m} n_i = n, \quad t \in [t_0, t_{END}],$$

with given vector valued parameter function $u(t)$ and unknown function $y(t)$.
In today's applications the system of DAE's can involve tens of thousands of equations or more. For solving such large–scale problems we consider a two level hierarchical structure of the system. The first level of the structure is build by the subsystems of the DAE–system. To exploit this structure for parallelization, it is assumed that for each subsystem the corresponding part of the function F_i as well as the hyperrows of the Jacobian $\partial F_i/\partial y + \alpha * \partial F_i/\partial \dot{y}$, with integration constant α (see [6]), can be computed independently of the other subsystems. The second level of the hierarchy is obtained by merging subsystems to blocks

$$\tilde{\mathcal{F}}_j = (F_{j_1}, F_{j_2}, \ldots, F_{j_{m_j}})^T, \quad j = 1(1)p, \quad \sum_{j=1}^{p} m_j = m. \tag{2}$$

There are different algorithms for generating a so called block partitioning (2). The algorithm used in this connection exploits only topological informations. It merges subsystems to nearly equal sized blocks while trying to minimize the number of coupling variables between blocks and to optimize the number of blocks in relation to the number of equations and the number of available concurrent processors. The implemented algorithm is reliable for general problems, it is of low complexity and therefore it is fast. An algorithm to generate a block partitioning with numerically weakly coupled blocks is described in [12, 13].
In Section 2 it is described how the hierarchical structure of the DAE–system can be used to construct effective parallelizable block–structured Newton–type methods. These methods require to solve many linear systems with the same pattern structure of the matrices but with different right hand sides. For it the Gaussian elimination method with pseudo code techniques is used. How this method can be adapted advantageously to vector and parallel computers is described in Section 3. Finally, results for large scale real life applications are given in Section 4.

2 Block–structured Newton–type methods

Based on a block partitioning (2) we extend the system of nonlinear equations arising from backward differentiation formulas (BDF) [6] to use Block–structured Newton–type methods for its solution on parallel computers. The extension is done by determining the internal variables $x = (x_1, \ldots, x_p)^T$ of the blocks, doubling of external couple variables $z = (z_1, \ldots, z_p)^T$, and appending linear identification equations $\mathcal{G}(z) = 0$, yielding the extended system

$$\begin{aligned} \mathcal{F}_j(x_j, z_j) &= 0, \quad j = 1(1)p, \\ \mathcal{G}(z) &= 0, \end{aligned} \tag{3}$$

where the nonlinear functions $\mathcal{F}_j$, corresponding to the blocks $\tilde{\mathcal{F}}_j$, have disjunctive arguments.
We split each block function $\mathcal{F}_j$, with $\mathcal{F}_j : \mathbb{R}^{r_j} \times \mathbb{R}^{s_j} \longrightarrow \mathbb{R}^{n_j}$, $r_j + s_j \geq n_j$, into $\mathcal{F}_j = (\mathcal{F}_j^1, \mathcal{F}_j^2)^T$ by determining r_j pivot elements in the $n_j \times r_j$ dimensional matrix $\partial_{x_j}\mathcal{F}_j := \partial\mathcal{F}_j/\partial x_j$. Thus the r_j pivot rows determine $\mathcal{F}_j^1$ (see Fig. 1) and it is ensured that $\partial_{x_j}\mathcal{F}_j^1$ is regular.

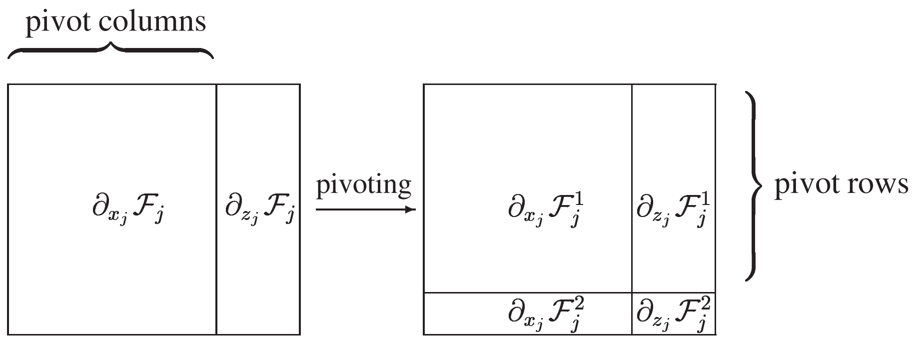

Fig. 1 Splitting of the equations $\mathcal{F}_j$.

Using the Newton–ansatz leads to

$$0 = \begin{pmatrix} \mathcal{F}_j^1 \\ \mathcal{F}_j^2 \end{pmatrix} + \begin{bmatrix} \partial_{x_j}\mathcal{F}_j^1 \\ \partial_{x_j}\mathcal{F}_j^2 \end{bmatrix} \Delta x_j + \begin{bmatrix} \partial_{z_j}\mathcal{F}_j^1 \\ \partial_{z_j}\mathcal{F}_j^2 \end{bmatrix} \Delta z_j, \quad j = 1(1)p, \tag{4}$$

$$0 = \mathcal{G} + \partial_z \mathcal{G} \Delta z, \tag{5}$$

with the following block structure of the Jacobian matrix:

$$\begin{bmatrix} \partial_{x_1}\mathcal{F}_1^1 & \partial_{z_1}\mathcal{F}_1^1 & & & & & \\ \partial_{x_1}\mathcal{F}_1^2 & \partial_{z_1}\mathcal{F}_1^2 & & & & & \\ & & \partial_{x_2}\mathcal{F}_2^1 & \partial_{z_2}\mathcal{F}_2^1 & & & \\ & & \partial_{x_2}\mathcal{F}_2^2 & \partial_{z_2}\mathcal{F}_2^2 & & & \\ & & & & \ddots & & \\ & & & & & \partial_{x_p}\mathcal{F}_p^1 & \partial_{z_p}\mathcal{F}_p^1 \\ & & & & & \partial_{x_p}\mathcal{F}_p^2 & \partial_{z_p}\mathcal{F}_p^2 \\ & \partial_{z_1}\mathcal{G} & & \partial_{z_2}\mathcal{G} & & & \partial_{z_p}\mathcal{G} \end{bmatrix}$$

Because the matrices $\partial_{x_j}\mathcal{F}_j^1, \;\; j = 1(1)p$, are regular, one gets

$$\Delta x_j = -\Delta\hat{x}_j - B_j \Delta z_j, \quad j = 1(1)p, \tag{6}$$

$$0 = \hat{\mathcal{F}}_j + C_j \Delta z_j, \qquad j = 1(1)p, \tag{7}$$

$$0 = \mathcal{G} + \partial_z \mathcal{G} \Delta z, \tag{8}$$

with

$$\Delta\hat{x}_j := (\partial_{x_j}\mathcal{F}_j^1)^{-1}\mathcal{F}_j^1, \quad \hat{\mathcal{F}}_j := \mathcal{F}_j^2 - \partial_{x_j}\mathcal{F}_j^2 \Delta\hat{x}_j,$$

$$B_j := (\partial_{x_j}\mathcal{F}_j^1)^{-1}\partial_{z_j}\mathcal{F}_j^1, \quad C_j := \partial_{z_j}\mathcal{F}_j^2 - \partial_{x_j}\mathcal{F}_j^2 B_j.$$

Thus the correction of the external variables Δz can be computed from the so called main system equations (7),(8) and the correction of the internal variables Δx can then be computed from the block system equations(6). Using the notations C :=diag(C_j) and $\hat{\mathcal{F}} := (\hat{\mathcal{F}}_1, \hat{\mathcal{F}}_2, \ldots, \hat{\mathcal{F}}_p)^T$ we can formulate the following basic algorithm for the evaluation of the corrections $\Delta x^{k+1} = x^{k+1} - x^k$ and $\Delta z^{k+1} = z^{k+1} - z^k$ in the k'th iteration step of a modified Newton method with scalar constant c:

(1) **do parallel** for all $j \in \{1, \ldots, p\}$:

(a) for new Jacobian: (i) compute the Jacobians $\partial_{x_j}\mathcal{F}_j$ and $\partial_{z_j}\mathcal{F}_j$ and do LU–factorization for $\partial_{x_j}\mathcal{F}_j$

(ii) solve:
$$\begin{bmatrix} \partial_{x_j}\mathcal{F}_j^1 & \emptyset \\ \partial_{x_j}\mathcal{F}_j^2 & I \end{bmatrix} \begin{bmatrix} B_j \\ C_j \end{bmatrix} = \begin{bmatrix} \partial_{z_j}\mathcal{F}_j^1 \\ \partial_{z_j}\mathcal{F}_j^2 \end{bmatrix} \tag{9}$$

(b) compute the function $\mathcal{F}_j(x_j^k, z_j^k)$ and solve:
$$\begin{bmatrix} \partial_{x_j}\mathcal{F}_j^1 & \emptyset \\ \partial_{x_j}\mathcal{F}_j^2 & I \end{bmatrix} \begin{pmatrix} \Delta\hat{x}_j^{k+1} \\ \hat{\mathcal{F}}_j^{k+1} \end{pmatrix} = c \begin{pmatrix} \mathcal{F}_j^1(x_j^k, z_j^k) \\ \mathcal{F}_j^2(x_j^k, z_j^k) \end{pmatrix} \tag{10}$$

enddo

(2) **do sequential**

(a) for new Jacobian: do LU–factorization of main system matrix $[C, \partial_z\mathcal{G}]^T$

(b) solve:
$$\begin{bmatrix} C \\ \partial_z\mathcal{G} \end{bmatrix} \Delta z^{k+1} = - \begin{pmatrix} \hat{\mathcal{F}}^{k+1} \\ c\,\mathcal{G}(z^k) \end{pmatrix} \tag{11}$$

enddo

(3) **do parallel** for all $j \in \{1, \ldots, p\}$: $\Delta x_j^{k+1} = -\Delta\hat{x}_j^{k+1} - B_j \Delta z_j^{k+1}$ **enddo**

Both main parts of the computational amount, namely all the calculation of functions and Jacobians and most of the amount for the solution of the linear systems, are included together in one parallel loop (step 1) resulting into a coarse grain parallelism. The bottleneck is the sequential part (step 2) which is dominated by the LU–factorization of the main system matrix. To reduce this sequential amount of the algorithm and increase the efficiency of the implementation on parallel computers various modifications of the method can be considered.

At first, the main system factorization can be speeded up by eliminating zero elements (*sparsing*) from the dense computed submatrix blocks C_j. For instance for a DAE system of a distillation plant with 13 436 equations using 14 blocks the resulting 1 138 equations of the main system comprise 67 342 matrix elements which can be reduced to 24 504 nonzero elements.

At second, by using *multilevel Newton iteration* techniques [3, 17] it is possible to shift computational costs from the main system solution (outer iteration) to the solution of the blocks by substituting step (1)(b) by an inner iteration loop:

set $\hat{x}_j^{k+1,0} := x_j^k$

do $l = 0(1)l_j$ **:** compute $\mathcal{F}_j(x_j^{k+1,l}, z_j^k)$ and solve:
$$\begin{bmatrix} \partial_{x_j}\mathcal{F}_j^1 & \emptyset \\ \partial_{x_j}\mathcal{F}_j^2 & I \end{bmatrix} \begin{pmatrix} \Delta\hat{x}_j^{k+1,l+1} \\ \hat{\mathcal{F}}_j^{k+1,l+1} \end{pmatrix} = c \begin{pmatrix} \mathcal{F}_j^1(x_j^{k+1,l}, z_j^k) \\ \mathcal{F}_j^2(x_j^{k+1,l}, z_j^k) \end{pmatrix} \tag{12}$$
and set $\hat{x}_j^{k+1,l+1} := \hat{x}_j^{k+1,l} + \Delta\hat{x}_j^{k+1,l+1}$

enddo

set $\Delta\hat{x}_j^{k+1} := \Delta\hat{x}_j^{k+1,l_j+1}$ and $\hat{\mathcal{F}}_j^{k+1} := \hat{\mathcal{F}}_j^{k+1,l_j+1}$

At third, by formally extending the main system (11) with the last factorized block–diagonal matrix $\hat{C}$,

$$\begin{bmatrix} C+\hat{C}-\hat{C} \\ \partial_z\mathcal{G} \end{bmatrix} \Delta z^{k+1} \;=\; -\begin{pmatrix} \hat{\mathcal{F}}^{k+1} \\ c\,\mathcal{G}(z^k) \end{pmatrix},$$

a new factorization of the main system matrix can be avoided, in some cases, by using the following *iterative scheme for solving the main system* instead of step (2)(b):

set $z_j^{k+1,0} := z_j^k,\ j = 1(1)p$
do $l = 0(1)l_0$ **:**
do parallel for all $j \in \{1,\dots,p\}$

$$\begin{bmatrix} \partial_{x_j}\mathcal{F}_j^1 & \emptyset \\ \partial_{x_j}\mathcal{F}_j^2 & I \end{bmatrix} \begin{pmatrix} B_j\Delta z_j^{k+1,l} \\ (C_j-\hat{C}_j)\Delta z_j^{k+1,l} \end{pmatrix} \;=\; \begin{pmatrix} \partial_{z_j}\mathcal{F}_j^1\Delta z_j^{k+1,l} \\ (\partial_{z_j}\mathcal{F}_j^2-\hat{C}_j)\Delta z_j^{k+1,l} \end{pmatrix} \tag{13}$$

enddo

$$\begin{bmatrix} \hat{C} \\ \partial_z\mathcal{G} \end{bmatrix} \Delta z^{k+1,l+1} \;=\; -\begin{pmatrix} \hat{\mathcal{F}}_2^{k+1} + (C-\hat{C})\Delta z^{k+1,l} \\ c\,\mathcal{G}(z^{k,l}) \end{pmatrix} \tag{14}$$

enddo
set $\Delta z^{k+1} := \Delta z^{k+1,l_0+1}$

Step (3) can then be replaced by:

do parallel for all $j \in \{1,\dots,p\}$: $\Delta x_j^{k+1} \;=\; -\Delta\hat{x}_j^{k+1} - B_j\Delta z_j^{k+1,l_0}$ **enddo**

In (12) as well as in (13) and (14) only existing factorizations are used and the corrections in (13) can be computed in parallel for all blocks $j = 1(1)p$. The integer constants $l_j, j = 0(1)p$ depend on the convergence properties of the method.

3 Gaussian elimination method with pseudo code

For solving systems of linear equations

$$Ax = b, \quad A \in \mathbb{R}^{n\times n}, \quad x, b \in \mathbb{R}^n,$$

with unsymmetric and sparse matrices A, we use the Gaussian elimination method

$$PAQ \;=\; LU, \tag{15}$$
$$Ly = Pb, \qquad UQ^{-1}x = y. \tag{16}$$

L is a lower triangular matrix and U an upper triangular matrix. The row permutation matrix P is used to provide numerical stability and the column permutation matrix Q is used to control sparsity. In the following, two cases for the determination of P and Q are considered.

In the first case, we determine in each elimination step a permutation of Q and then a permutation of P. With it also the columns are reordered dynamically. The permutation of Q is done by using the first column of the eliminated matrix with a minimal number of nonzero elements as the pivot column [13]. For keeping the method numerically stable at stage k of the elimination, the pivot $a_{i,j}$ is selected among those candidates satisfying the numerical threshold criterion

$$|a_{i,j}| \geqq \beta \max_{l \geqq k} |a_{l,j}|, \tag{17}$$

with a threshold parameter $\beta \in (0, 1]$.
In the second case, we find the permutation matrix Q by minimum degree ordering of $A^T A$ or of $A^T + A$. Then, in a separate step we determine P using the numerical threshold criterion (17).
To perform several factorizations of PAQ into the product of L and U for matrices A with the same pattern structure and the same pivot sequence as well as to solve $Ly = Pb$ and $UQ^{-1}x = y$ for several right hand sides, a pseudo code is generated. This code describes the operations that are necessary to refactorize A with (15) and to resolve the linear systems (16). It can be formulated independently of a computer [12]. For vectorization as well as for parallelization one has to find elements of LU just as x and y respectively that can be computed independently of each other. Using the algorithm of Yamamoto and Takahashi [18] a level of independence is defined for each matrix element of LU in (15) and each vector element of x and y in (16), so that elements with the same level number can be computed concurrently. Based on this information, one can define vector statements for the pseudo code instructions (vectorization) or spread the pseudo code instructions among different processors (parallelization).
Let $a(i)$ denote the nonzero elements in LU. Then for vector computers the following vector statements have been proven to be successful for the factorization:

$$\begin{aligned}
s &= \sum_{\kappa} a(i_\kappa) * a(j_\kappa), \\
a(i_k) &= 1/a(i_k), \\
a(i_k) &= a(i_k) * a(i_l), \\
a(i_k) &= (a(i_l) * a(i_m) + a(i_p) * a(i_q)) * a(i_k),
\end{aligned}$$

The array elements are addressed indirectly because of the representation of LU as a sparse matrix.
For parallel computers we distribute the pseudo code for each level of independence to about equal parts on the processors. After the processors have executed their part of the pseudo code instructions of a level concurrently, a synchronization among the processors is needed, before the execution of the next level can be started.

4 Applications

Both the nonlinear and the linear solver have been proven successfully for dynamic process simulation of real life chemical plants. They are now included in a block oriented process simulation code BOP using block partitioning algorithms and numerical integration with BDF methods [6]. It uses a structured data interface, which is currently generated [15] out of the data supplied by the commercial process simulator SPEEDUP [1] [1]. The interface describes the system of DAE's structured into subsystems corresponding to the units of the plant and is usable for an independent evaluation of subsystem functions and Jacobian matrices. Our simulation code BOP is currently implemented on moderate parallel computers Cray J90 with up to 32 processors and on a SGI Origin 2000 with 8 processors. Carrying out dynamic process simulations for large–scale distillation plants of the Bayer AG Leverkusen have shown the potential parallelism of the methods realized in BOP (see Tables 1 and 2).

Table 1. Dynamic simulation of plant *bayer11* (190 subsystems, 10 226 equations)

Processors	1	1	4	8	12
Blocks	1	24	24	24	24
Coupling variables	0	500	500	500	500
$\sum$ CPU time (in seconds)	694.90	730.96	855.37	865.72	839.65
Wall clock time (in seconds)	701.37	732.12	245.33	160.72	116.21
Speedup factor	1	0.96	2.86	4.35	6.04

Table 2. Dynamic simulation of plant *bayer12* (170 subsystems, 19 558 equations)

Processors	1	1	7	8	12
Blocks	1	21	21	24	24
Coupling variables	0	819	819	897	897
$\sum$ CPU time (in seconds)	1 038.39	934.31	1 079.93	1 059.01	1 080.07
Wall clock time (in seconds)	1 040.60	936.04	230.31	219.82	175.96
Speedup factor	1	1.11	4.52	4.73	5.91

The times given in Table 1 and 2 are measured for whole simulation runs on non dedicated machines Cray J90 and include the times for sequential pre– and post–processing, which are usually about 5% of the overall CPU time. From a performance analysis using the Cray tool ATExpert it was found that for the dynamic simulation of plant *bayer12* using 16 processors and a 16 block partitioning a speedup factor of 11.5 is possible.

For some large matrices arising from circuit simulation and chemical process simulation (Table 3) our linear solver GSPAR has been compared with the solvers UMFPACK [7] and SuperLU [8]. Results for using GSPAR with dynamical column reordering and with minimum degree ordering of $A^T A$ or $A^T + A$, signed with ♦ and ★ respectively, are given in Table 4. In this table *op LU* denotes the number of operations for the factorization, *nz LU* the number of nonzero elements in L+U, and *strat* the CPU time (in seconds) on a DEC AlphaServer with a EV5.6 processor used for the first factorization including the analysis and the generation of pseudo code.

[1] Used under licence 95122131717 for free academic use from AspenTech

Table 3. Matrices for GSPAR, UMFPACK and SuperLU

Name	Discipline	Equations	Nonzero elements
advice3388	circuit	3 388	40 545
advice3776	simulation	3 776	27 590
meg1		2 904	58 142
meg4		5 960	46 842
bayer01	chemical	57 735	277 774
bayer04	engineering	20 545	159 082
bayer10		13 436	94 926

Table 4. GSPAR with dynamical column reordering and minimum degree ordering

	Dynamical column reordering			Minimum degree ordering		
Name	op LU	nz in LU	strat	op LU	nz in LU	strat
advice3388	310 348	49 842	1.03	396 965★	50 363	1.70
advice3776	355 465	53 246	1.10	382 224★	53 664	1.60
meg1	796 797	98 578	0.72	1 245 847★	117 700	1.26
meg4	420 799	85 626	1.30	376 324★	81 850	0.78
bayer01	10 032 621	921 672	47.90	13 860 173♦	1 090 279	15.70
bayer04	5 954 718	427 088	9.26	6 340 579♦	449 103	4.72
bayer10	5 992 500	322 601	5.60	3 953 687♦	298 559	2.68

Table 5 gives the CPU times (in seconds) needed for one refactorization of the matrices on a DEC AlphaStation with a EV4.5 processor. Here GSPAR was used with dynamical column reordering (dco) and minimum degree ordering (mmd) alternatively.

Table 5. Refactorization with GSPAR, UMFPACK, and SuperLU

	GSPAR		UMFPACK	SuperLU
Name	with dco	with mmd		
advice3388	0.10	0.12★	0.25	0.28★
advice3776	0.10	0.12★	0.30	0.42★
meg1	0.22	0.37★	0.58	1.43★
meg4	0.13	0.13★	0.37	0.75★
bayer01	3.18	4.62♦	5.02	6.70♦
bayer04	1.68	1.70♦	3.37	2.77♦
bayer10	1.63	1.02♦	1.60	1.57♦

For some large matrices arising from chemical process simulation we compared the vector version of our linear solver GSPAR with FAMP [19], the frontal method of SPEEDUP [1], and found that factorizations with GSPAR are up to two times faster on Cray vector computers [13]. Thus using GSPAR alternatively to FAMP in SPEEDUP it was possible to reduce the CPU–time for the whole dynamic simulation of large distillation plants to up to 62%.

The performance of the parallel version of GSPAR on a computer Cray T3D is shown in Table 6. For this example the pseudo code is to large to distribute it among up to 8 processors.

Table 6. Example bayer01: CPU time (in seconds) for refactorization on Cray T3D

Processors	CPU time	Speedup factor
16	2.36	set to 1.00
32	1.45	1.63
64	0.95	2.47

Acknowledgments. This work was supported by the German Federal Ministry of Education, Science, Research, and Technology under grant GA7FVB-3.0M370. The valuable assistance and the technical support from the Bayer AG Leverkusen, the Cray Research GmbH a Silicon Graphics company Munich, and the AspenTech UK Ltd Cambridge are gratefully acknowledged.

References

[1] AspenTech: *SPEEDUP, User Manual, Library Manual*, Aspen Technology, Inc., Cambridge, Massachusetts, USA, (1995).

[2] Bock, H. G., Schlöder, J.P., Schulz, V. H.: *Numerik großer Differentiell–Algebraischer Gleichungen – Simulation und Optimierung*, in: Prozeßsimulation (H. Schuler, Hrsg.), VCH Verlagsgesellschaft, Weinheim, S. 35–80 (1995).

[3] Borchardt, J., Grund, F., Horn, D., Uhle, M.: *MAGNUS – Mehrstufige Analyse großer Netzwerke und Systeme*, Tech. Report 9, WIAS, Berlin, (1994).

[4] Borchardt, J., Grund, F., Horn, D., Michael, T.: *Parallelized Numerical Methods for Large–Scale Dynamic Process Simulation*. In: Sydow, A. (ed.): Proceedings of the 15th IMACS World Congress on Scientific Computation, Modelling and Applied Mathematics, Wissenschaft & Technik Verlag, Berlin, Volume I, pp. 547–552 (1997).

[5] Brüll, L., Pallaske U.: *On Consistent Initialization of Differential–Algebraic Equations with Discontinuities*. In: Wacker, Hj., Zulehner W. (eds.): Proceedings of the Fourth Conference on Mathematics in Industry, pp. 213–217 (1991).

[6] Brenan, K.E., Campbell, S.L., Petzold, L.R.: *Numerical Solution of Initial–Value Problems in Differential–Algebraic Equations*, North–Holland, New York, (1989).

[7] Davis, T.A., Duff, I.S.: *A Combined Unifrontal/Multifrontal Method for Unsymmetric Sparse Matrices*. Technical Report TR97-016, CISE Dept., Univ. of Florida, Gainesville, (1997).

[8] Demmel, J.W., Eisenstat, S.C., Gilbert, J.R., Li, X.S., Liu, J.W.H.: *A supernodal approach to sparse partial pivoting*. Technical Report UCB//CSD-95-883, Computer Science Division, U.C. Berkeley, (1995).

[9] Elst, G., Pönisch, G.: *On two–level Newton methods for the analysis of large–scale nonlinear networks*. In: Proceedings of the ECCTD'85, Prag, pp. 165–169 (1985).

[10] Griepentrog, E., Hanke, M., März, R.: *Toward a better understanding of differential algebraic equations (Introductory survey)*, Seminarberichte Nr. 92–1, Humboldt–Universität zu Berlin, Fachbereich Mathematik, (1992).

[11] Grund F.: *Numerische Lösung von hierarchisch strukturierten Systemen von Algebro–Differentialgleichungen*. In: Bank, R., Burlirsch, R., Gajewski, H., Merten, K. (eds.): Modelling and Simulation of Electrical Circuits and Semiconductor Devices, International Series of Numerical Mathematics, vol. 111, Birkhäuser Verlag, Basel, S. 17–31 (1994).

[12] Grund, F., Borchardt, J., Horn, D., Michael, T., Sandmann, H.: *Differential–algebraic systems in the chemical process simulation*. In: Keil, F., Mackens, W., Voss, H., Werther, J. (eds.): Scientific Computing in Chemical Engineering, Springer–Verlag, Berlin, pp. 68–74 (1996).

[13] Grund, F., Michael, T., Brüll, L., Hubbuch, F., Zeller, R., Borchardt, J., Horn, D., Sandmann, H.: *Numerische Lösung großer strukturierter DAE–Systeme in der chemischen Prozeßsimulation*. In: Hoffmann, K.H., Jäger, W., Lohmann, Th., Schunk, H. (Hrsg.): Mathematik–Schlüsseltechnologie für die Zukunft, Springer–Verlag, Berlin Heidelberg, S. 91–103 (1997).

[14] Gräb, R., Günther, M., Wever, U., Zheng, Q.: *Optimization of Parallel Multilevel–Newton Algorithms on Workstation Clusters*. In: L. Bouge et al. (Eds.): Euro–Par96 Parallel Processing, Berlin, Lecture Notes in Computer Science 1124, Springer-Verlag, Berlin Heidelberg, p. 91–96 (1996).

[15] Horn, D.: *Entwicklung einer Schnittstelle für einen DAE–Solver in der chemischen Verfahrenstechnik*. In: Mackens, W., Rump, S.M. (Hrsg.),: Software Engineering im Scientific Computing, Vieweg & Sohn Verlagsgesellschaft mbH, Braunschweig, S. 249–255 (1996).

[16] Hoyer, W., Schmidt, J.W.: *Newton–Type Decomposition Methods for Equations Arising in Network Analysis*, ZAMM 64, No. 9, pp. 397–405 (1984).

[17] Rabbat, N.B.G., Sangiovanni–Vincentelli, A.L., Hsieh, H.Y.: *A Multilevel Newton Algorithm with Macromodeling and Latency for the Analysis of Large-Scale Nonlinear Cirquits in the Time Domain*, IEEE Transactions on CAD of Integrated Cicuits and Systems, vol. CAD–1, No.3, pp. 131–145 (1982).

[18] Yamamoto, F., Takahashi, S.: *Vectorized LU decomposition algorithms for large–scale circuit simulation*, IEEE Transactions on Computer Aided Design of Integrated Cicuits and Systems, CAD–4, pp. 231–239 (1985).

[19] Zitney, S.E., Stadtherr, M.A.: *Frontal algorithms for equation–based chemical process flowsheeting on vector and parallel computers*, Computers & Chemical Engineering, 17, pp. 319–338 (1993).

Authors' adress: J. Borchardt, F. Grund and D. Horn, Weierstrass Institute for Applied Analysis and Stochastics, Mohrenstrasse 39, D – 10117 Berlin, Federal Republic of Germany.