Weierstraß–Institut
für Angewandte Analysis und Stochastik

im Forschungsverbund Berlin e.V.

Preprint ISSN 0946 – 8633

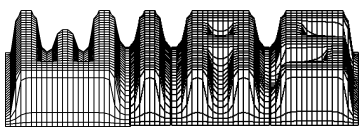
Parallel Numerical Methods for Large-Scale DAE Systems

Jürgen Borchardt

submitted: 29th February 2000

Weierstrass Institute
for Applied Analysis
and Stochastics
Mohrenstrasse 39
10117 Berlin
Germany
E-Mail: borchard@wias-berlin.de

Preprint No. 558
Berlin 2000



1991 Mathematics Subject Classification. 65Y05,80A30,65L05,65H10,65F50.

Key words and phrases. Systems of differential–algebraic equations, Block partitioned systems, Newton–type methods, Sparse–matrix techniques, Parallelization, Chemical process simulation, Dynamic simulation of distillation plants.

This work was supported by the Federal Ministry of Education, Science, Research and Technology, Germany under grant GR7FV1.

Edited by
Weierstraß–Institut für Angewandte Analysis und Stochastik (WIAS)
Mohrenstraße 39
D — 10117 Berlin
Germany

Fax: + 49 30 2044975
E-Mail (X.400): c=de;a=d400-gw;p=WIAS-BERLIN;s=preprint
E-Mail (Internet): preprint@wias-berlin.de
World Wide Web: http://www.wias-berlin.de/

Abstract

For plantwide dynamic simulation in chemical process industry, parallel numerical methods using a divide and conquer strategy are considered. An approach for the numerical solution of initial value problems for large systems of differential algebraic equations (DAEs) arising from industrial applications and its realization on parallel computers with shared memory is discussed. The system is partitioned into blocks and then it is extended appropriately, such that block-structured Newton-type methods can be applied which enable the application of relaxation techniques. This approach has gained considerable speedup factors for the dynamic simulation of various large-scale distillation plants, covering systems with up to 60 000 equations.

1 Introduction

During the last three decades, dynamic process simulation has become an indispensable tool for process design and operation in chemical industry. Particularly due to an improved accuracy of mathematical process models and an increasing degree of integration in process modeling, the size of the problems which have to be solved numerically has grown considerably within this time. With that, the plantwide dynamic process simulation has become an challenging field of application for parallel numerical methods.

For a dynamic process simulation of complex, highly interconnected plants, initial value problems for large-scale systems of coupled differential and algebraic equations (DAEs)

$$\begin{aligned} F(t, y(t), \dot{y}(t), u(t) = 0, \qquad & t \in [t_0, t_{end}], \\ y(t_0) = y_0, & \\ \dot{y}(t_0) = \dot{y}_0, & \end{aligned}$$

with given piecewise continuous parameter function *u(t)* and the unknown function *y(t)*, have to be solved. Generally, differential equations arise from balances of energy, mass, and momentum, while algebraic

relations result from constitutive relations for phenomenological quantities or different constraints, for example. These DAE systems are highly nonlinear and can involve several tens of thousands of equations or even more. Because the following is focussed on the parallelization aspect of numerical methods only, for the considerations in this paper it is assumed that the DAE system is of index 1 (Brenan *et al.*, 1989) and that consistent initial values are provided (Kröner *et al.*, 1992). Although the DAE systems in chemical process simulation are usually linearly implicit, the general implicit representation will be used for notational simplicity.

The goal of this paper is to show, how the hierarchical, modular structure of chemical plants can be exploited for large-scale dynamic process simulation on parallel computers. For that a plant is considered as a network of connected process units like reactors, pumps, heat exchangers, or trays of distillation columns. In an equation based flowsheeting approach, a parameter dependent mathematical model, describing the unit operations, is assigned to each unit type and the units are connected, e.g. by mass and energy streams. With it the corresponding system of DAEs can be structured into subsystems according to the units.

In Section 2, a short overview on parallel numerical methods for DAE systems is given, to classify the parallel methods considered in this paper. A simulation concept based on divide and conquer techniques is introduced in Section 3. It uses the given unit structure of chemical plants to get hierarchically structured, block-partitioned systems of DAEs. Section 4 describes how this hierarchical structure can be used to construct efficiently parallelizable block-structured Newton-type methods. These methods, formally based on block Schur-complement techniques, require a repeated solution of linear systems with the same pattern structure of sparse, unsymmetric coefficient matrices and with multiple right-hand sides. A direct solver with pseudo code techniques is used to solve these linear systems. Most operations of block-structured Newton-type methods can be performed independently for the blocks. Particularly all the calculations of functions and Jacobian matrices as well as most of the amount needed for the solution of the linear systems can be covered together in one parallel loop. This results in a coarse-grained parallelism. Finally, in Section 5, the implementation of the methods in a prototype of the block-oriented process simulation code BOP and results for the dynamic simulation of large-scale real world applications of Bayer AG in Leverkusen on a parallel computer with shared memory are discussed.

2 Approaches to Parallelization

Since the DAE systems which occur in real world chemical process simulation usually represent problems with multiple time scales, implicit integration methods, e.g. BDF methods (Brenan *et al.*, 1989), are used for their solution. The resulting systems of nonlinear algebraic equations, which have to be solved at each discrete point of time, are then treated with quadratically convergent Newton-type methods. Hence, systems of linear algebraic equations with sparse coefficient matrices have to be solved for each Newton step.

Since this matrices are in general structurally unsymmetric and are not diagonally dominant, preferably direct methods are used for their solution. A parallelization of numerical methods for initial value problems for DAE systems is in general possible at all three levels of the numerical solution process,

- at the level of **DAEs**,
- at the level of **nonlinear equations**, as well as
- at the level of **linear equations**.

The possible granularity of the parallelism increases within this level hierarchy from linear equations to DAEs. For all levels, most of the parallel approaches are based on a *partitioning* of the system into subsidiary systems. This partitioning is cohered with an *assignment* of variables to equations or subsystems, respectively.

As the solution of large sparse **systems of linear equations** is usually the most computationally intensive part in large-scale dynamic process simulation, a parallelization approach at this level is quite obvious. Two examples for such an approach should be mentioned here. A generally applicable approach that works independently of a partitioning of the system of linear equations is described by Grund (1999). Here pseudo code techniques are used for vectorization as well as for parallelization. The pseudo code instructions describing the operations for LU-factorization and solution of the sparse linear system are assorted successively into groups of operations, so that the operations in each group can be performed independently, after the operations of the previous group have been performed. Thus, a vectorization or parallelization approach can be established by realizing for each group the formation of vector operations or the balanced distribution of the operations to the processors, respectively. In another approach, Mallya *et al.* (1997) describe a parallel linear solver based on a multifrontal technique. Here the frontal elimination is performed simultaneously in multiple independent or loosely connected blocks. For this method a partitioning of the linear system corresponding to a bordered block-diagonal form of the coefficient matrix is needed. Camarda and Stadtherr (1999) propose a graph partitioning algorithm that reorders rows and columns so that such a structure with similarly sized diagonal blocks and a possibly small interface matrix can be obtained. In general, one of these or any other parallel linear solver can be adapted with less effort to nearly any dynamic simulation package. Thus, an easy to realize, universally applicable parallelization approach for dynamic process simulation can be established, if the parallelism is restricted to the linear solver. In opposite to that, parallel approaches at the higher levels of the numerical solution process enlarge the scope for parallelization, but they usually require an appropriately adapted simulation concept.

The most rigorous parallel approach can be implemented at the level of the **DAE system**. If this system can be partitioned into several weakly coupled, lower-ordered systems (blocks), then dynamic iteration methods, so called *waveform relaxation* (WR) methods can be applied. These methods, originated from Picard iteration, have been first successfully applied to VLSI circuit simulation (see Lelarasmee *et al.*, 1982). Within the iteration process of a waveform relaxation method, the blocks of the DAE system are solved independently over time horizons, so called windows. This enables a *multirate integration* by using

different step-sizes and orders for different subsystems of DAEs. The implementation of WR methods results in a highly parallelizable concurrent fraction and low sequential overhead. To our knowledge, WR methods have been first applied to chemical process simulation by Skellum et all. (1988) and Secci et. all. (1991,1993). While in general for the block partitioning of systems of linear equations the coupling between blocks needs to be only topologically weak, the partitioning of the DAE system into numerically weakly coupled blocks based on a consistent assignment of variables to equations is necessary (see Lelarasmee et all.,1982). Grund et all. (1996) proposed an improved assignment and partitioning algorithm and Michael and Borchardt (1996) proved convergence of the WR method for a simplified tray model. Nevertheless, convergence problems may occur in real world applications or it can be hard to find a suitable partitioning. Because of that, WR methods currently seem to be no all purpose methods for the dynamic process simulation of homogeneously modeled plants, but in one or the other way they need to be considered in the face of a heterogeneous multiple plant simulation (see Paloschi and Zitney, 1999). For that, Ehrhardt *et al.* (1999) sketch a quasi-Newton acceleration of the waveform relaxation method using a block Broyden update approach for approximating the dynamic sensitivities of submodels.

For the homogeneous plantwide dynamic process simulation, a parallelization approach at the level of the **systems of nonlinear equations** can be seen as a medium between the two approaches sketched before. It will be shown in the following, that based on an easily obtainable partitioning, a generally applicable parallel approach covering “exact” methods as well as relaxation techniques can be established at this level.

3 A Unit-Oriented Divide and Conquer Strategy

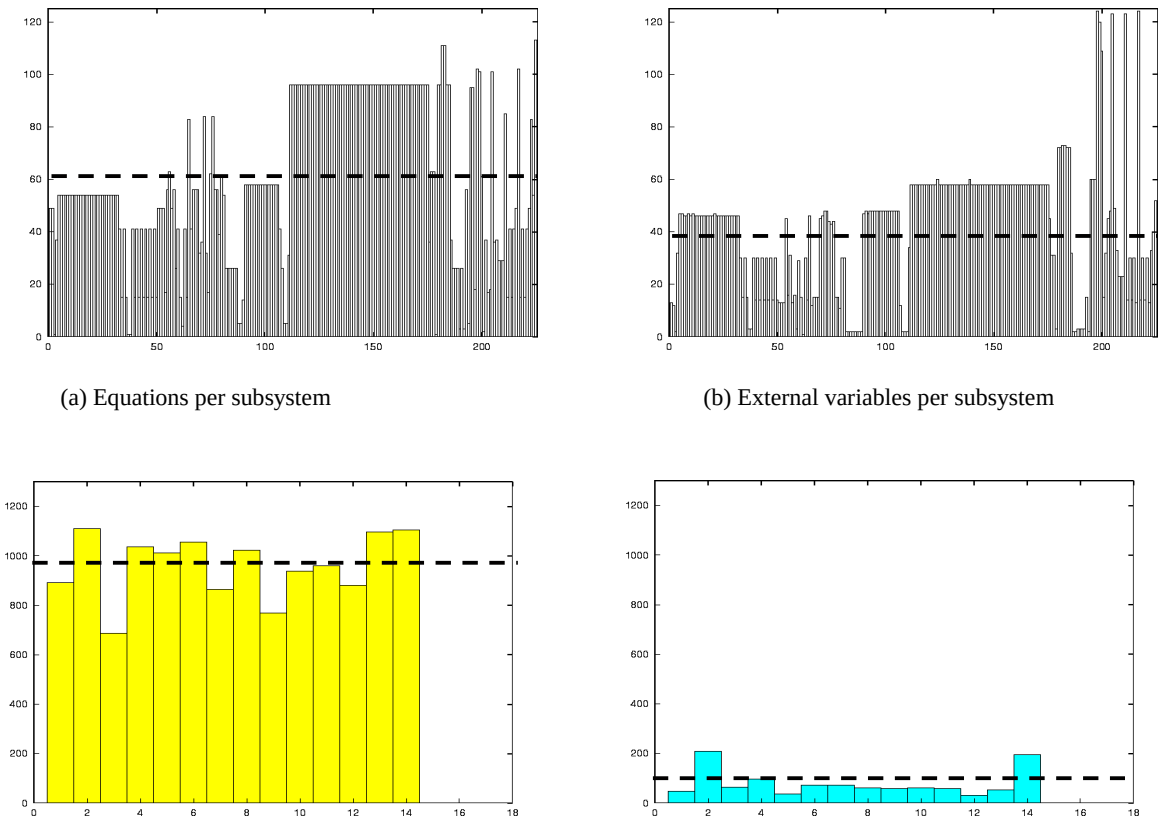
Due to a modular modeling of unit operations in equation-based process flowsheeting, the DAE systems can be structured into *m* coupled subsystems

$$\begin{aligned} &F_i(t, y(t), \dot{y}(t), u(t) = 0,\ i = 1(1)m, \\ &F_i : \mathbb{R} \times \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^l \to \mathbb{R}^{n_i},\ \sum_{i=1}^{m} n_i = n, t \in [t_0, t_{end}]. \end{aligned} \tag{1}$$

Here each subsystem corresponds to a unit of the plant. Based on this structure, divide and conquer techniques can be used in a hierarchical simulation concept which is suitable for an implementation on parallel computers. For that, a two level hierarchical structure of the DAE system is considered. The first level of the structure is built by the subsystems of the DAE system. It is assumed that for each subsystems the corresponding parts of the function as well as of the Jacobian matrix can be computed independently of the analogous parts of the other subsystems. The second level of the hierarchy is obtained by merging subsystems to blocks

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

Such a so called block partitioning (2) of the DAE system can be predefined, e.g. by a macro model description covering functional blocks, or can otherwise be generated automatically by different partitioning algorithms. For the methods described in the next section, a very simple algorithm using a heuristic approach has proven to be sufficient in almost all cases. It uses only topological information generated from the generally known nonzero structure of the Jacobian of (1). In contrast to other graph partitioning algorithms in this field, it does not operate on the level of equations or variables respectively, but on the level of subsystems. That means, subsystems can only be merged to form blocks and can not be split.



(a) Equations per subsystem

(b) External variables per subsystem

(c) Equations per block

(d) External variables per block

Fig. 1 Block partitioning statistics for a distillation plant (14 blocks, 225 subsystems, 13 436 equations)

It is the main goal of the algorithm to minimize the number of overall external coupling variables between blocks while getting as far as possible similarly sized blocks. As it will be shown in the next section, the number of overall coupling variables between blocks can be viewed as a measure for the sequential computational amount of the methods to be constructed and so it mainly determines the maximally possible speedup due to Amdahl's law. At the other hand, a similar block size usually will give a good load balance. The simplest partitioning strategy following this goal can roughly be described as follows. In each partitioning step, the subsystem or block with the highest ratio of external variables per equations is merged with the subsystem or block that has as many as possible external variables in common with the

first one and for which the number of equations of the resulting block does not exceed a threshold value. This threshold value is updated dynamically from one partitioning step to the next. The main termination condition of the algorithm is fulfilled if a given number of blocks is reached. This final block number can be preset or can be computed from the number of subsystems, the number of equations, and the number of available processors. Another termination condition is fulfilled, if the average ratio of external variables per equations of the blocks is fallen below a given bound. The resulting block partitioning can be appraised and some statistics can be viewed. In Figure 1, part of such a block partitioning statistics for a distillation plant with 225 units is viewed. The resulting DAE system has 13 436 equations and 225 subsystems, where each subsystem corresponds to a unit. It can be seen that without block partitioning, i.e. each subsystem forms a block, the average rate of external variables per subsystem is approximately two third of the average rate of equations per subsystem (Figures 1a, 1b). For a block partitioning into 14 blocks the average rate of external variables per block is reduced to less then one tenth of the average rate of equations per block and the block sizes do not differ too much (Figures 1c, 1d), although in this case the main weight in the partitioning algorithm has been on minimizing the number of coupling variables .
Based on the sketched partitioning, a unit-oriented simulation concept can be applied for a parallelization approach at the level of nonlinear algebraic equations.

4 Parallel Newton-type methods

If implicit integration methods, e.g. BDF methods (Brenan *et al.*, 1989), are used for solving the initial value problems (1), then a system of nonlinear equations

$$\tilde{\mathcal{F}}(y) = 0,$$

has to be solved at each discrete point of time. On the basis of a block partitioning (2), this system can be formally extended to use block-structured Newton-type methods for its solution on parallel computers. The extension is done by determining the internal variables $x = (x_1, x_2, \ldots, x_p)^T$ of the blocks, duplicating of external variables to form $z = (z_1, z_2, \ldots, z_p)^T$, and appending identification equations. The external coupling variables enclose "original" external variables as well as their duplicated counterparts. This approach yields the extended block partitioned system

$$\mathcal{F}_j(x_j, z_j) = 0, \qquad j = 1(1)p, \tag{3a}$$

$$\mathcal{G}(z) = 0, \tag{3b}$$

where the nonlinear functions $\mathcal{F}_j : \mathbb{R}^{r_j} \times \mathbb{R}^{s_j} \to \mathbb{R}^{q_j}$, $r_j \le q_j \le r_j + s_j$, corresponding to the blocks $\tilde{\mathcal{F}}_j$ in (2) have disjunctive arguments and the function $\mathcal{G} : \mathbb{R}^s \to \mathbb{R}^{r+s-n}$, with

$$\sum_{j=1}^{p} r_j = r, \quad \sum_{j=1}^{p} s_j = s, \text{ and } \sum_{j=1}^{p} q_j = n,$$

is linear. With the notations

$$\Delta x_j^{k+1} := x_j^{k+1} - x_j^k, \quad \partial_{x_j} \mathcal{F}_j := \frac{\partial \mathcal{F}_j}{\partial x_j}(x_j^k, z_j^k),$$

and corresponding terms for $\Delta z_j^{k+1}, \partial_{z_j} \mathcal{F}_j$ and $\partial_{z_j} \mathcal{G}$, the equations to be solved in the *k*th iteration step of a Newton-type approach for the extended system (3) are of the form

$$0 = \mathcal{F}_j(x_j^k, z_j^k) + \partial_{x_j} \mathcal{F}_j \Delta x_j^{k+1} + \partial_{z_j} \mathcal{F}_j \Delta z_j^{k+1}, \qquad j = 1(1)p, \tag{4a}$$

$$0 = \mathcal{G}(z^k) + \sum_{j=1}^{p} \partial_{z_j} \mathcal{G} \Delta z_j^{k+1}. \tag{4b}$$

The Jacobian matrix of the extended system has a block structure, which is formally shown in Figure 2.

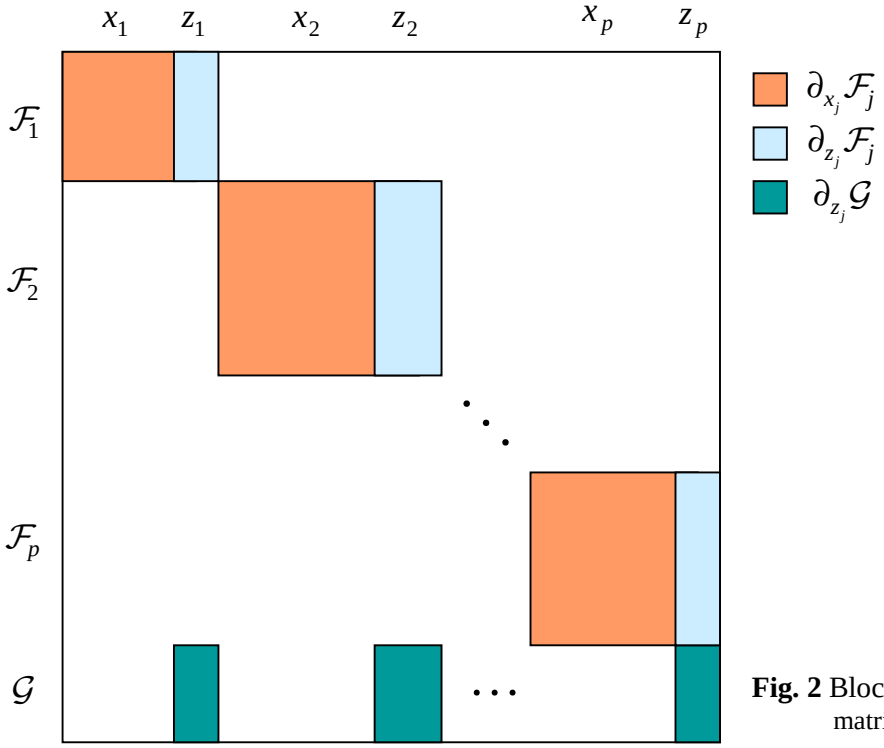


Fig. 2 Block structure of the Jacobian matrix of (3)

This formal extension of the system of nonlinear equations can be exploited for different parallelization approaches.

4.1 Universally applicable block-structured Newton-type methods

Borchardt (1998) and Borchardt *et al.* (1999a) propose universally applicable block-structured Newton-type methods based on a splitting of the block functions $\mathcal{F}_j$ into $\mathcal{F}_j = (\mathcal{F}_j^1, \mathcal{F}_j^2)^T$. Using the modified pivoting algorithm of a linear solver, the splitting is obtained by determining r_j pivot elements in the $q_j \times r_j$ dimensional matrices $\partial_{x_j} \mathcal{F}_j$, so that the pivot rows determine $\mathcal{F}_j^1$ (see Figure 3.). This splitting can change during the numerical integration of (1) to ensure the regularity of $\partial_{x_j} \mathcal{F}_j^1$ e.g. after discontinuities.

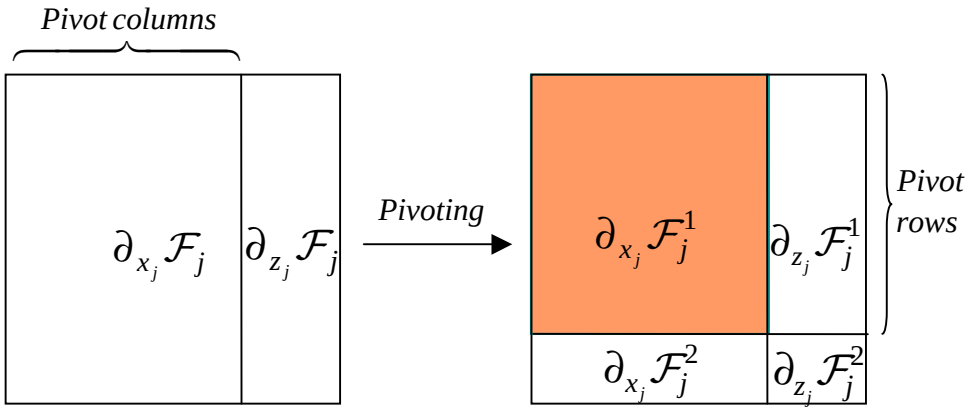


Fig. 3 Splitting of the equations $\mathcal{F}_j$

Since the matrices $\partial_{x_j}\mathcal{F}_j^1,\ j=1(1)p,$ are regular by construction, one gets from (4)

$$\Delta x_j^{k+1} = -\Delta\hat{x}_j^{k+1} - B_j\Delta z_j^{k+1}, \qquad j=1(1)p, \tag{5a}$$

$$0 = \hat{\mathcal{F}}_j^{k+1} + C_j\Delta z_j^{k+1}, \qquad j=1(1)p, \tag{5b}$$

$$0 = \mathcal{G}(z^{k+1}) + \partial_z\mathcal{G}\Delta z^{k+1}, \tag{5c}$$

with the abbreviations

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

It is obvious that, if the first approximates $\Delta\hat{x}_j^{k+1}$ for the corrections of the internal variables, the right-hand sides $\hat{\mathcal{F}}_j^{k+1}$, as well as the sensitivity matrices B_j, and the block Schur-complement matrices C_j have been computed for all blocks, then the corrections of the external variables Δz^{k+1} can be computed from the coupling equations (5b),(5c), forming the so called main system. At last, the corrections of the internal variables Δx^{k+1} can then be computed from the block equations (5a).

So, forming the block diagonal matrix $C := \mathrm{diag}(C_j)$ and collecting the right-hand sides $\hat{\mathcal{F}}_j^{k+1}$ to form $\hat{\mathcal{F}}^{k+1}$, one gets from (5) that the evaluation of the corrections $\Delta x^k = (\Delta x_1^k, \Delta x_2^k, \cdots, \Delta x_p^k)^T$ and $\Delta z^k = (\Delta z_1^k, \Delta z_2^k, \cdots, \Delta z_p^k)^T$ in the *k*th iteration step of a modified Newton method can be efficiently realized in the following basic steps:

step 1: do parallel for all $j \in \{1,2,..., p\}$:

(a) for new Jacobian: (i) compute the coefficient matrix in (6*a*)
and generate its LU-factorization

(ii) solve:

$$\begin{bmatrix} \partial_{x_j}\mathcal{F}_j^1 & 0 \\ \partial_{x_j}\mathcal{F}_j^2 & I \end{bmatrix}\begin{bmatrix} B_j \\ C_j \end{bmatrix} = \begin{bmatrix} \partial_{z_j}\mathcal{F}_j^1 \\ \partial_{z_j}\mathcal{F}_j^2 \end{bmatrix} \tag{6a}$$

(b) compute the function $\mathcal{F}_j(x_j^k, z_j^k)$ and solve:

$$\begin{bmatrix} \partial_{x_j}\mathcal{F}_j^1 & 0 \\ \partial_{x_j}\mathcal{F}_j^2 & I \end{bmatrix} \begin{pmatrix} \Delta\hat{x}_j^{k+1} \\ \hat{\mathcal{F}}_j^{k+1} \end{pmatrix} = c \begin{pmatrix} \mathcal{F}_j^1(x_j^k, z_j^k) \\ \mathcal{F}_j^2(x_j^k, z_j^k) \end{pmatrix} \tag{6b}$$

enddo

step 2: do sequential

(a) for new Jacobian: generate the LU-factorization of the main system matrix in (6*c*)

(b) solve:

$$\begin{bmatrix} C \\ \partial_z \mathcal{G} \end{bmatrix} \Delta z^{k+1} = - \begin{pmatrix} \hat{\mathcal{F}}^{k+1} \\ c\,\mathcal{G}(z^k) \end{pmatrix} \tag{6c}$$

enddo

step 3: do parallel for all $j \in \{1,2,..., p\}$:

$$\Delta x_j^{k+1} = -\Delta\hat{x}_j^{k+1} - B_j \Delta z_j^{k+1} \tag{6d}$$

enddo

For the choice of the scalar constant *c* of the modified Newton method it is referred to Brenan *et al.* (1989). To solve (6*a*) and (6*b*) the same LU-factorizations are used several times. Consequently, the linear solver should be particularly efficient in solving linear systems with multiple right-hand sides. Beside this, the solver has to be generalized for pivoting rectangular matrices to obtain the splitting of the block functions and it should exploit the special structure of the coefficient matrix for factorization.

In this paper Newton-type methods based on (6*a*)-(6*d*) are called *Type 1* methods. For these methods, the operations in **step 1**, and **step 3** can be done concurrently for all block systems. Implementing them on parallel computers with shared memory, both main parts of the computational amount, namely all the calculation of functions and Jacobian matrices as well as most of the amount for the solution of the linear systems, can be covered together in one parallel loop built up from **step 1**. This results in a coarse-grained parallelism. The bottleneck of parallelization is the sequential **step 2**, which is dominated by the LU-factorization of the main system matrix.

To reduce the sequential computational amount of the algorithm and to increase the efficiency of the implementation on parallel computers, various modifications of the method can be introduced. First of all, zero elements as well as very small elements of the computed dense sub-matrix blocks C_j can be eliminated by *sparsing* techniques. With that, the number of nonzero elements in the main system matrix can be reduced to less then 30% for large-scale problems. That decreases the time needed for pivoting of the main system matrix considerably.

Another well known modification are so *called multilevel Newton iteration* techniques (Rabbat et all., 1982). Here it is tried to shift computational costs from the main system solution (outer iteration) to the solution of the block systems 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 $\mathcal{F}_j(\hat{x}_j^{k+1,l}, z_j^k)$ and solve:

$$\begin{bmatrix} \partial_{x_j}\mathcal{F}_j^1 & 0 \\ \partial_{x_j}\mathcal{F}_j^2 & I \end{bmatrix} \begin{pmatrix} \Delta\hat{x}_j^{k+1,l+1} \\ \hat{\mathcal{F}}_j^{k+1,l+1} \end{pmatrix} = c \begin{pmatrix} \mathcal{F}_j^1(\hat{x}_j^{k+1,l}, z_j^k) \\ \mathcal{F}_j^2(\hat{x}_j^{k+1,l}, z_j^k) \end{pmatrix} \qquad (6b')$$

$$\hat{x}_j^{k+1,l+1} := \hat{x}_j^{k+1,l} - \Delta\hat{x}_j^{k+1,l+1}$$

enddo

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

Usually only 2 or 3 iterations of the inner loop are performed per outer iteration.

Finally, modifications of block-structured Newton-Type methods can be obtained by introducing *relaxation* techniques for the solution of the main system equations. The main goal of this approach is to reduce the number of main system factorizations. If for example, the main system (6*c*) is formally extended to:

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

where $\bar{C}$ denotes the old block-diagonal matrix from the last factorized main system matrix, then, using a Jacobi-type relaxation approach, a new factorization of the main system matrix can be avoided by replacing **step 2** by the following iterative scheme:

Set $z_j^{k+1,0} := z_j^k$, $\bar{\mathcal{F}}_j^{k+1,0} = \emptyset$, $j = 1(1)p$,

do $l = 0(1)l_0$:

solve:

$$\begin{bmatrix} \bar{C} \\ \partial_z\mathcal{G} \end{bmatrix} \Delta z^{k+1,l+1} = -\begin{pmatrix} \hat{\mathcal{F}}^{k+1} + \bar{\mathcal{F}}^{k+1,l} \\ c\,\mathcal{G}(z^k) \end{pmatrix} \qquad (6c')$$

do parallel for all $j \in \{1,2,..., p\}$:

solve:

$$\begin{bmatrix} \partial_{x_j}\mathcal{F}_j^1 & 0 \\ \partial_{x_j}\mathcal{F}_j^2 & I \end{bmatrix} \begin{pmatrix} \Delta\bar{x}_j^{k+1,l+1} \\ \bar{\mathcal{F}}_j^{k+1,l+1} \end{pmatrix} = \begin{pmatrix} \partial_{z_j}\mathcal{F}_j^1 \Delta z_j^{k+1,l+1} \\ (\partial_{z_j}\mathcal{F}_j^2 - \bar{C}_j)\Delta z_j^{k+1,l+1} \end{pmatrix} \qquad (6a')$$

enddo

enddo

set $\Delta z_j^{k+1} := \Delta z_j^{k+1,l_0+1}$, $\Delta\bar{x}_j^{k+1} := \Delta\bar{x}_j^{k+1,l_0+1}$.

In this modification, the approximation of the defect by using the old factorization instead of the new one is expressed by a modification of the right-hand side in $(6c')$, where $\bar{\mathcal{F}}^{k+1,l}$ is formed by the $\bar{\mathcal{F}}_j^{k+1,l}$ of all blocks $j = 1(1)p$. The explicit evaluation of the matrices B_j and C_j is avoided by computing only the matrix-vector products $\Delta\bar{x}_j^{k+1,l+1} = B_j \Delta z_j^{k+1,l+1}$ and $\bar{\mathcal{F}}_j^{k+1,l+1} = (C_j - \bar{C}_j)\Delta z_j^{k+1,l+1}$ from $(6a')$. That means that the solution of (6*a*) in **step 1** can be skipped in this case and (6*d*) has to be substituted by

$$\Delta x_j^{k+1} = -\Delta\hat{x}_j^{k+1} - \Delta\bar{x}_j^{k+1}. \qquad (6d')$$

Hence, all operations except of the solution of $(6c')$ can now be performed in parallel for the blocks.

4.2 Block-structured Newton-type methods with input / output specifications

Another possibility to reduce the computational amount for the solution of the coupling system is based on identifying input and output streams of the units in the flowsheet. If according to this, the external variables z_j of the block systems can be divided into input variables u_j and output variables v_j, with

$$z_j = (u_j, v_j)^T, \quad u_j \in \mathbf{R}^{r_j+s_j-q_j}, v_j \in \mathbf{R}^{q_j-r_j},$$

then the linear function $\mathcal{G}$ can be chosen such that $\partial_v \mathcal{G} = -I$ and $\Delta v = \partial_u \mathcal{G} \Delta u$. Here, *s=2(n-r)* is assumed for notational simplicity. With it, $\partial_u \mathcal{G}$ is a permutation matrix and there is only one input per output, but the following can be extended to the multiple input case as well. The resulting special block structure of the extended nonlinear system (3) is formally shown in Figure 4.

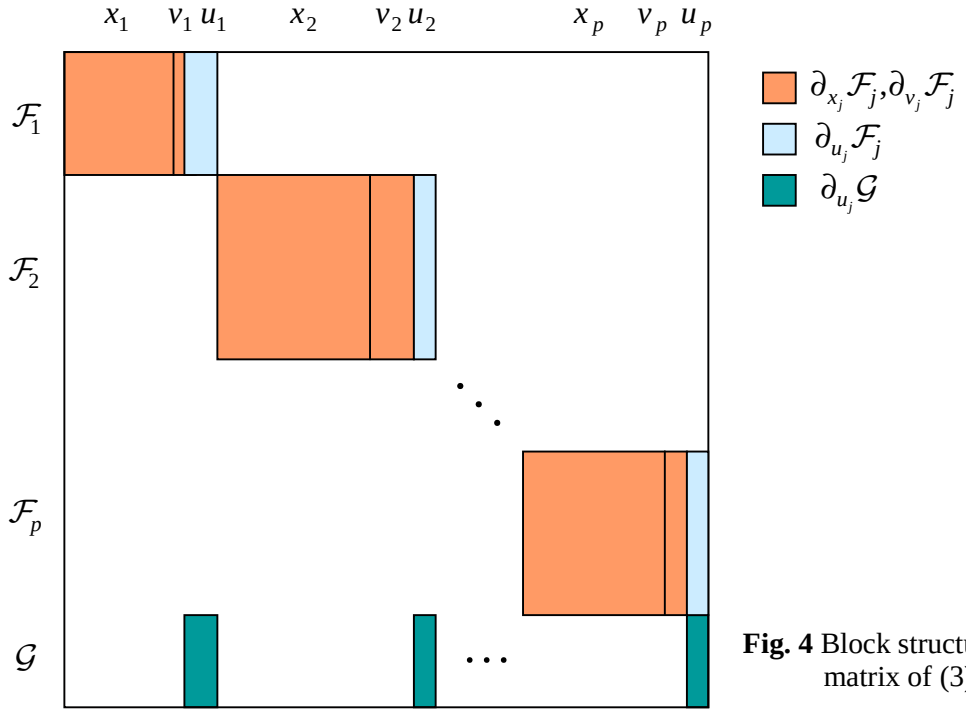


Fig. 4 Block structure of the Jacobian matrix of (3) for *Type 2* methods

If the output variables can be computed from the block system equations together with the internal variables, then an inverse

$$\left[\partial_{x_j}\mathcal{F}_j \quad \partial_{v_j}\mathcal{F}_j\right]^{-1}$$

exists and it is not necessary to split the block system equations. Using the abbreviation $B_v = diag(B_{v_j})$, this results into the basic algorithm for *Type 2* methods:

step 1: do parallel for all $j \in \{1,2,...,p\}$:

(a) for new Jacobian: (i) compute the coefficient matrix in (7*a*)
and generate its LU-factorization
(ii) solve:

$$\left[\partial_{x_j}\mathcal{F}_j \quad \partial_{v_j}\mathcal{F}_j\right]\begin{bmatrix} B_{x_j} \\ B_{v_j} \end{bmatrix} = \partial_{u_j}\mathcal{F}_j \tag{7a}$$

(b) compute the function $\mathcal{F}_j(x_j^k, u_j^k, v_j^k)$ and solve:

$$\left[\partial_{x_j}\mathcal{F}_j \quad \partial_{v_j}\mathcal{F}_j\right]\begin{pmatrix} \Delta\hat{x}_j^{k+1} \\ \Delta\hat{v}_j^{k+1} \end{pmatrix} = c\mathcal{F}_j(x_j^k, u_j^k, v_j^k) \tag{7b}$$

enddo

step 2: do sequential

(a) for new Jacobian: generate the LU-factorization of the main system matrix in (7*c*)

(b) solve:

$$\begin{aligned} \left[\partial_u\mathcal{G} + B_v\right]\Delta u^{k+1} &= -\Delta\hat{v}^{k+1}, \\ \Delta v^{k+1} &= \partial_u\mathcal{G}\Delta u^{k+1} \end{aligned} \tag{7c}$$

enddo

step 3: do parallel for all $j \in \{1,2,...,p\}$:

$$\Delta x_j^{k+1} = -\Delta\hat{x}_j^{k+1} - B_{x_j}\Delta u_j^{k+1} \tag{7d}$$

enddo

Compared to *Type 1* methods, where only the internal variables have been assigned to the block equations, here the internal and the output variables are assigned to the blocks. With that, the size of the main system is reduced to the half. Of course, modifications as *sparsing*, *multilevel Newton iteration*, and *relaxation* techniques can also be used in the *Type 2* case. Apart from the fact that *Type 1* methods can be applied to more general problems, *Type 2* methods enable a better parallelization and applicability of relaxation techniques.

In the case of weakly coupled blocks, a relaxation based modification of a *Type 2* method can be introduced by using previous values of Δu to approximate $B_v\Delta u$ in (7*c*). For a Jacobi-type relaxation approach, the sequential **step 2** can then be substituted by

Set $\Delta v_j^{k+1,0} := -\Delta \hat{v}_j^{k+1}, \quad j = 1(1)p,$

do $l = 0(1)l_0$:

$$\Delta u^{k+1,l} = \left[\partial_u \mathcal{G}\right]^T \Delta v^{k+1,l}$$

do parallel for all $j \in \{1,2,..., p\}$:

solve:

$$\left[\partial_{x_j}\mathcal{F}_j \quad \partial_{v_j}\mathcal{F}_j\right]\begin{pmatrix}\Delta \bar{x}_j^{k+1,l+1}\\ \Delta \bar{v}_j^{k+1,l+1}\end{pmatrix} = \partial_{u_j}\mathcal{F}_j \Delta u_j^{k+1,l}, \tag{7a'}$$

$$\Delta v_j^{k+1,l+1} = -\Delta \hat{v}_j^{k+1} + \Delta \bar{v}_j^{k+1,l+1} \tag{7c'}$$

enddo

enddo

set $\Delta v_j^{k+1} := \Delta v_j^{k+1,l_0+1}, \ j = 1(1)p, \ \Delta u^{k+1} = \left[\partial_u \mathcal{G}\right]^T \Delta v^{k+1}.$

Here again, the explicit evaluation of the sensitivity matrices B_{x_j} and B_{v_j} of the internal and of the output variables with respect to input variables is avoided by computing only the vectors

$$\Delta \bar{x}_j^{k+1,l+1} = B_{x_j} \Delta u_j^{k+1,l}, \ \Delta \bar{v}_j^{k+1,l+1} = B_{v_j} \Delta u_j^{k+1,l}.$$

Because of this, the operations for the solution of (7*a*) in **step 1** can be skipped as well, if (7*d*) is substituted by

$$\Delta x_j^{k+1} = -\Delta \hat{x}_j^{k+1} - \Delta \bar{x}_j^{k+1}. \tag{7d'}$$

Except of the permutation operations, all other operations of such a relaxation modification of a *Type 2* method can be done in parallel.

For monitoring convergence of the relaxation iteration, one should look to what the replacement of (7*c*) with (7*c*') means. Because of

$$D\Delta u = D(\partial_u \mathcal{G})^{-1}\partial_u \mathcal{G}\Delta u = D(\partial_u \mathcal{G})^T \Delta v = \tilde{D}\Delta v,$$

with $D = \partial_u \mathcal{G} + B_v$ and $\tilde{D} = I + B_v(\partial_u \mathcal{G})^T$, the solution of (7*c*) is equivalent to solving

$$\left[I + B_v(\partial_u \mathcal{G})^T\right]\Delta v^{k+1} = -\Delta \hat{v}$$

and (7*c*') is equivalent to

$$\Delta v^{k+1,l+1} = -\Delta \hat{v} - B_v(\partial_u \mathcal{G})^T \Delta v^{k+1,l}.$$

The transformation of the matrix D into $\tilde{D}$ is formally shown in Figure 5. Convergence can now be monitored by using the quantities $\left\| B_{v_j} \right\|_1$ for example.

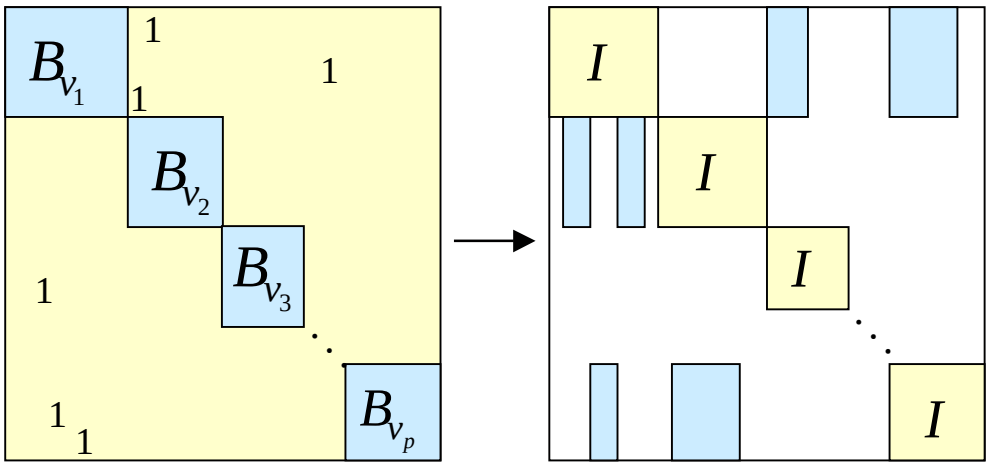


Fig. 5 Transformation of the main system matrix for *Type 2* methods

Successive substitution appears as a special case of this relaxation modification of *Type 2* methods, if all sensitivity matrices B_{x_j} and B_{v_j} are set to zero. That also implies the possibility that only some of the sensitivity matrices are updated. Because these sensitivity matrices just represent that kind of information that is neglected in ordinary waveform relaxation approaches, *Type 2* methods can be used to study the convergence problems that can appear for WR methods. Beside that, the block-structured Newton-Type methods also offer the possibility to exploit occasionally unequal activity (latency) of units.

5 Implementation and Results

Starting from a unit-based hierarchical modeling, a parallel-modular approach for the dynamic simulation of large-scale chemical processes is implemented in the prototype of the **B**lock **O**riented **P**rocess simulation package BOP. The key methods for parallelization are currently the block-structured Newton-type methods described in Section 4. BOP uses a hierarchically structured data interface (Horn, 1996). The interface describes the system of DAEs as structured into subsystems corresponding to the units of the plant and is usable for an independent evaluation of subsystem functions and Jacobian matrices. Until now it has been generated from the data supplied by the process simulator SPEEDUP (Aspen Technology, Inc., 1998), but currently a first version of a compiler for a purpose-designed, high-level modeling language for process simulation, both developed at Weierstrass Institute, is tested. Thus, the data interface for BOP can upcoming be generated from a problem description with this highly structured language. A tool for transforming a process description with this language to a subset of the language used in SPEEDUP and vise versa already exists. A comparable tool with respect to the simulator DIVA (Kröner, 1990) is planned. Currently the data interfaces of DIVA and BOP can be generated mutually. With the modeling language of BOP also a predefined block partitioning can be described. If no block decomposition is predefined, a topological block partitioning algorithm is used within BOP. It merges subsystems to blocks and reports

partitioning statistics. Due to it's low complexity it is very fast. Usually, a block partitioning is generated only once during preprocessing, but it is also possible to generate a new one during dynamic simulation.
Currently, only BDF methods are included to apply implicit numerical integration to the usually stiff systems of DAEs. For that the DASSL code from Petzold (1991) has been modified by replacing the nonlinear and linear solver, adding the handling of discontinuities, and introducing changes in consistent initialization (Kröner *et al.*, 1992), and error tests. The systems of nonlinear equations are solved with the block-structured Newton-type methods. This methods require the repeated solution of linear systems. For solving these systems with unsymmetric and sparse coefficient matrices, an extended version of the linear solver GSPAR (Grund, 1999) is covered. It uses the Gaussian elimination method. The nonzero elements of the coefficient matrix are stored in compressed sparse row format. To control sparsity and to ensure numerical stability, a pivoting algorithm with dynamic reordering of columns and partial pivoting in the pivot columns is used. Pivoting strategies with different numerical complexity are implemented. Pseudo code instructions are generated to perform several factorizations for matrices with the same pattern structure using the same pivot sequence as well as to solve the linear system for multiple right-hand sides. This instructions describe the operations that are necessary at one hand for the factorization and at the other hand for the solution of the linear system. Using the pseudo code technique enables a fast refactorization as well as an efficient handling of multiple right-hand sides. It can be exploited for vectorization by grouping independent operations of the same type to vector operations. For using GSPAR within BOP, GSPAR has been extended for pivoting on matrices for which the number of rows exceeds the number of columns (see Figure 3 in Section 3) and for exploiting the special structure of the matrices for block-structured Newton methods of *Type 1* during factorization (see (6*a*),(6*b*) in Section 4).
The simulation package BOP is currently implemented on shared memory computers Cray J90, SGI Origin 2000 and Compaq AlphaServer using multiprocessing compiler directives for parallelization (see OpenMP, 1999). Used for the dynamic process simulation of various large distillation plants of Bayer AG in Leverkusen, BOP has shown a good parallel performance. Some results are presented below. All times given in Table 1 to Table 5 have been measured for whole simulation runs on non dedicated computers Cray J90 and include the times for sequential pre- and post-processing.

Processors	1	1	7	8	21
Blocks	1	21	21	8	21
Coupling variables	0	819	819	273	819
CPU (sec.)	1 250	1 124	1 161	1 306	1 142
Wall clock (sec.)	1 285	1 148	245	292	142
Speedup factor	1.00	1.12	5.24	4.40	9.05

Table 1. Dynamic simulation with BOP for plant **bayer12** (170 units, 19 558 equations)

Processors	1	1	4	12	24
Blocks	1	24	24	24	24
CPU (sec.)	860	904	937	920	900
Wall clock (sec.)	877	920	271	124	89
Speedup factor	1.00	0.95	3.24	7.07	9.80

Table 2. Dynamic simulation with BOP for plant **bayer14** (190 units, 13 436 equations)

Processors	1	1	8	12	24
Blocks	1	24	24	24	24
CPU (sec.)	606	639	717	675	676
Wall clock (sec.)	622	654	147	116	85
Speedup factor	1.00	0.95	4.23	5.36	7.32

Table 3. Dynamic simulation with BOP for plant **bayer13** (296 units, 18 350 equations)

Processors	1	1	1	8
Blocks	1	16	32	16
CPU (sec.)	1 833	1 071	809	1 538
Wall clock (sec.)	1 866	1 084	825	372
Speedup factor	1.00	1.72	2.26	5.02

Table 4. Dynamic simulation with BOP for plant **bayer01** (785 units, 57 735 equations)

In Table 1 the number of coupling variables, which is related to the size of the main system, is listed additionally. For the treated examples, a blocksize of about one thousand was found to be efficient, but that should depend on the complexity of the unit models and the degree of integration in the process modeling. The results for example **bayer01** have to be discussed under the constraint that it was not possible to perform function as well as Jacobian evaluations for the blocks concurrently for this example. That was due to common blocks (save variables) used in some library procedures that have been not accessible. Therefore for this example, the operations for function and Jacobian evaluations had to be protected by a guarded region or critical section inside the parallel region, what allows only one processor at a time to do work of this region.

In Table 5 the performance of BOP using different block-structured Newton-type methods is compared to that of SPEEDUP on a Cray J90. The example is a reactor model built up modularly by a multiphase cell model which might be associated to a simplified reactive separation volume element. As expected, *Type 2* methods have a minor sequential overhead and parallelize better then *Type 1* methods.

Simulation with	Processors	Blocks	CPU (sec.)	Wall clock (sec.)
SPEEDUP	1	1	7 008	7 516
BOP	1	1	5 089	5 120
BOP with Type 1	1	18	5 814	5 870
BOP with Type 2	1	18	4 932	4 967
BOP with Type 1	6	18	6 208	1 904
BOP with Type 2	6	18	5 140	1 371

Table 5. Dynamic simulation for reactor600 (45 600 equations) on a CrayJ90

To see which performance could be possible on a dedicated computer, a performance analysis with the Cray tool *ATExpert* has been carried out. A result for the example **bayer12** can be seen in Figure 6. Here a 16-block partitioning was used to estimate the maximal speedup that can be expected for using up to 16 processors.

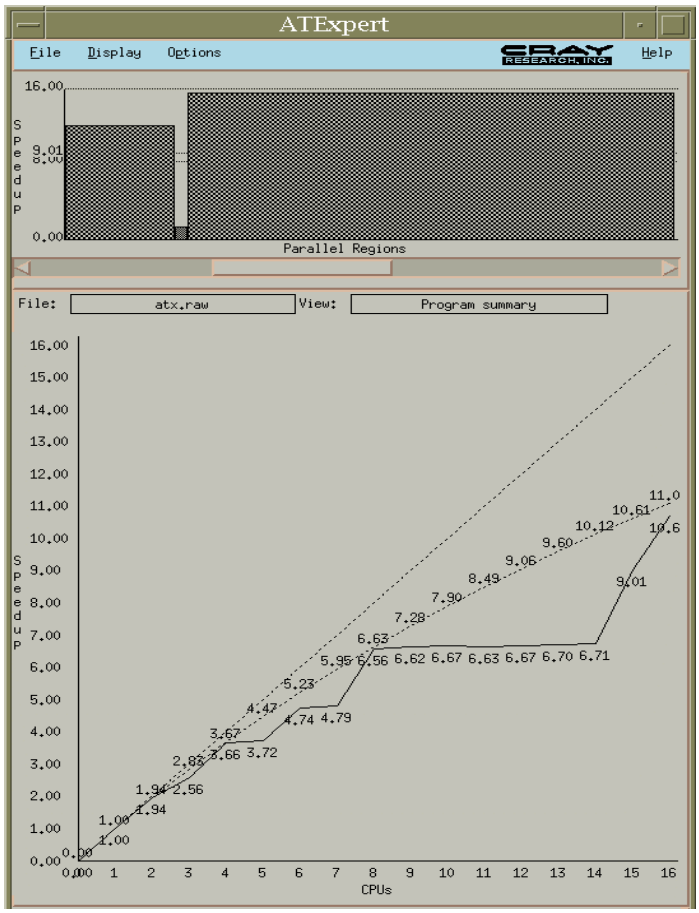


Fig. 6 Performance analysis with *ATExpert* for the dynamic simulation of **bayer12** on a CrayJ90, using BOP with a 16-block partitioning

6 Concluding Remarks

In this paper, parallel approaches to the numerical solution of DAE systems in large-scale dynamic process simulation have been discussed. Block-structured Newton-type methods are currently our favorite candidate for parallelization in homogeneously modeled plantwide dynamic process simulation. For these methods simple partitioning algorithms based on a unit-oriented modeling can be used to generate the data structure for a hierarchical simulation concept. The methods have shown to be generally applicable, reliable and numerically stable. Convergence problems can be avoided, because it can be switched from the relaxation modifications to an "exact" method at any point of the dynamic simulation. Since both main parts of the computational amount for solving the DAE system are covered together in one parallel region, a coarse grained parallelism is obtained. Often, even a sequential speedup is gained for large-scale problems. The implementation of a unit-oriented hierarchical simulation concept within the simulator BOP has proven to be successful for large-scale real world dynamic simulation applications on parallel computers with shared memory. A parallelization approach of block-structured Newton-type methods on distributed memory machines seems to be promising.

Acknowledgements

This work has been supported by the German Federal Ministry of Education and Research under grant GR7FV1. The author acknowledges the support of Bayer AG in Leverkusen, and thanks Aspen Technology, Inc. for providing a free SPEEDUP license for academic use. He is deeply indebted to his colleagues from Weierstrass Institute, Friedrich Grund, Dietmar Horn and Klaus Ehrhardt for the inspiring collaboration on the BOP project.

References

Aspen Technology Inc. (1997) SPEEDUP, Release 5.5-6, User Manual, Cambridge, Massachusetts.

Borchardt, J., (1998) Parallelized Block-Structured Newton-Type Methods in Dynamic Process Simulation. In: B. Kågström et al. (eds.), Applied Parallel Computing, Proceedings of PARA'98, Lecture Notes in Computer Science 1541, Springer-Verlag, Berlin, pp. 38-42.

Borchardt, J., Gärtner, K., Grund, F., Horn, B., Horn, D., Zielas, W. (1985) Basic principles of the network analysis program ANNET. In: Proceedings of the 7th European Conference on circuit theory and design, Prague, pp. 113-116.

Borchardt, J., Grund, F., Horn, D., Michael, T. (1997) Parallelized numerical methods for large-scale dynamic process simulation. In: Sydow, A. (ed.): Proceedings of the 15th IMACS World Congress on Scientific Computation, Modeling and Applied Mathematics, Wissenschaft & Technik Verlag, Berlin, vol. I, pp. 547-552.

Borchardt, J., Grund, F., Horn, D. (1999a) Parallelized Methods for Large Nonlinear and Linear Systems in the Dynamic Simulation of Industrial Applications. Surv. Math. Ind. 8, pp. 201-211.

Borchardt, J., Ehrhardt, K. Grund, F., Horn, D. (1999b) Parallel Modular Dynamic Process Simulation. In: Keil, F., Mackens, W., Voss, H., Werther, J. (eds.): Scientific Computing in Chemical Engineering II, Springer-Verlag, Berlin, pp. 152-159.

Brenan, K.E., Campbell, S.L., Petzold, L.R. (1989) Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations. North-Holland, New York.

Brüll, L. (1996) Simulation manager: A software tool for process simulation in a heterogeneous hard- and software environment. SPEEDUP Journal 10, pp. 51-53.

Brüll, L., Pallaske, U. (1991) 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.

Camarda, K.V., Stadtherr, M.A. (1999) Matrix ordering strategies for process engineering: graph partitioning algorithms for parallel computation. Comput. Chem. Engng. 23, pp. 1063-1073.

Ehrhardt, K., Borchardt, J., Grund, F., Horn, D. (1999) Divide and Conquer Strategies in Large Scale Dynamic Process Simulation. Comput. Chem. Engng. 23, Suppl., pp.335-338.

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

Grund, F., Borchardt, J., Horn, D., Michael, T., Sandmann, H. (1996) 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.

Grund, F. (1999) Direct Linear Solvers for Vector and Parallel Computers. In: Palma, J. M., Dongarra, J., Hernández, V. (eds.): Vector and Parallel Processing – VECPAR'98, Lecture Notes in Computer Science 1573, Springer-Verlag, Berlin, pp. 114-127.

Horn, D.(1996) Entwicklung einer Schnittstelle für einen DAE-Solver in der chemischen Verfahrenstechnik. In: Mackens, W., Rump, S.M. (Hrsg.), Software Engineering in Scientific Computing, Viehweg & Sohn, Braunschweig, pp. 249-255.

Hoyer, W., Schmidt, J.W. (1984) Newton-Type Decomposition Methods for Equations Arising in Network Analysis. ZAMM 64, pp. 397-405.

Kahlert, M., Paffrath, M., Wever, U. (1984) Grid Partitioning Versus Domain Decomposition: A Comparison in Several Industrial Problems. Surv. Math. Ind. 6, pp. 133-166.

Kröner, A., Marquardt, W., Gilles, E.D. (1992) Computing Consistent Initial Conditions for Differential Algebraic Equations. Computers & Chemical Engineering 16, pp. 131-138.

Kröner, A., Holl, P., Marquardt, W., Gilles, E.D. (1990) DIVA – An Open Architecture for Dynamic Simulation. Computers & Chemical Engineering 14, pp. 1289-1295.

Lelarasmee, E., Ruehli, A.E., Sangiovanni-Vincentelli, A.L. (1982) The Waveform Relaxation Method for Time-Domain Analysis of Large Scale Integrated Circuits. IEEE Transactions on CAD of Integrated Circuits and Systems 1, pp. 131-145.

Mallya, J.U., Zitney, S.E., Choudhary, S., Stadtherr, M.A. (1997) A parallel frontal solver for large scale process simulation and optimization. AIChE Journal, 43, pp. 1032-1040.

Mallya, J.U., Zitney, S.E., Choudhary, S., Stadtherr, M.A. (1999) Matrix reordering effect on a parallel frontal solver for large scale process simulation.. Comput. Chem. Engng. 23, pp. 585-593

Michael, T., Borchardt, J. (1996) Convergence Criteria for Waveform Iteration Methods Applied to Partitioned DAE Systems in Chemical Process Simulation. Preprint 262, WIAS, Berlin, pp. 1-19.

OpenMP (1998) Application Program Interface, http://www.openmp.org

Paloschi, J.R., Zitney, S.E. (1999) Parallel Dynamic Simulation of Industrial Chemical Processes on Distributed-Memory Computers. Comput. Chem. Engng. 23, Suppl., pp.395-398.

Petzold, L. (1991) DASSL, Differential Algebraic System Solver, http://www.netlib.org/ode/dassl.f

Rabbat, N.B.G., Sangiovanni-Vincentelli, A.L., Hsieh, H.Y. (1982) A Multilevel Newton Algorithm with Macro-modeling and Latency for the Analysis of Large-Scale Nonlinear Circuits in the Time Domain. IEEE Transactions Circuits and Systems 26, pp. 733-741.

Secchi, A.R., Laganier, F.S., Morari, M. (1991) Dynamic Process Simulation Using a Concurrent Differential and Algebraic Solver. European Symposium on Computer Aided Process Engineering-2, pp. 467-472.

Secchi, A.R., Morari, M., Biscaia Jr., E.C. (1993) The Waveform Relaxation Method in the Concurrent Dynamic Process Simulation. Computers & Chemical Engineering 17, pp. 683-704.

Skjellum, A. Morari, M., Mattison, S.. (1988) Waveform relaxation for concurrent dynamic simulation of distillation columns. In: Proc. Third Conf. On Hypercube Concurrent Computers and Applications, Vol. 2, Pasadena, pp. 683-704.

White, J.K., Sangiovanni-Vincentelli, A.L. (1987) Relaxation Techniques for the Simulation of VLSI-Circuits. Kluwer Academic Press, New York.

Yang, G., Dutto, L.C., Fortin, M. (1997) Inexact block Jacobi-Broyden methods for solving nonlinear systems of equations. SIAM J. Sci. Comput. 18, pp. 1367-1392.

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