Weierstraß-Institut
für Angewandte Analysis und Stochastik

im Forschungsverbund Berlin e.V.

Preprint ISSN 0946 – 8633

Solution of Linear Systems with Sparse Matrices

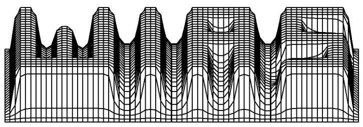
Friedrich Grund

Weierstrass Institute for Applied Analysis and Stochastics

E-Mail: grund@wias-berlin.de

submitted: 17th February 2003

No. 816
Berlin 2003



2000 *Mathematics Subject Classification.* Primary 65F05; Secondary 65F50, 65Y05.

Key words and phrases. Direct Linear Solver, Sparse–Matrix–Techniques, Parallel Computation, Circuit Simulation, Chemical Process Simulation.

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

Fax:	+ 49 30 2044975
E-Mail:	preprint@wias-berlin.de
World Wide Web:	http://www.wias-berlin.de/

Solution of Linear Systems with Sparse Matrices

Friedrich Grund

To my grandchildren Anna, Janina and Paula

Abstract. For large scale problems in electric circuit simulation as well as in chemical process simulation, the linear solver often needs about 50 – 80 % of the total amount of computing time. For that purpose, we consider direct methods for the numerical solution of linear systems of equations with unsymmetric sparse coefficient matrices. The Gaussian elimination method

$$\begin{aligned} PAQ &= LU, \\ Ly = Pb, \qquad & UQ^{-1}x = y \end{aligned}$$

is applied to solve the linear system $Ax = b$. Here, the row permutation matrix P is used to provide numerical stability and the column permutation matrix Q is chosen to control sparsity. In a new approach, implemented in the solver GSPAR2, the determination of the pivot columns is done with a modified algorithm, which has only a complexity of $O(n)$. A partial pivoting technique is used to maintain numerical stability.

For solving several linear systems with the same pattern structure of the coefficient matrix efficiently, we generate a list of pseudo code instructions for the factorization of the matrices. With it, the solver GSPAR2 has been proven successful within the simulation of several real life problems. For a number of linear systems arising from different technical problems, the computing times of GSPAR2 are compared to that of some recently released linear solvers.

1. Introduction

For the dynamic simulation of large scale applications in electric circuit analysis [2] as well as in chemical process engineering [3], initial value problems for large systems of nonlinear differential–algebraic equations (DAEs) have to be solved. For solving such problems, implicit integration methods like e.g. backward differentiation formulas [11] are used and the resulting systems of nonlinear equations

Received by the editors December 15, 2002.

1991 *Mathematics Subject Classification.* Primary 65F05; Secondary 65F50, 65Y05.

Key words and phrases. Direct Linear Solver, Sparse–Matrix–Techniques, Parallel Computation, Circuit Simulation, Chemical Process Simulation.

This work was supported by the Federal Ministry of Education and Research of the Federal Republic of Germany under grants GA7FVB and GR7FV1.

are then solved with Newton-type methods [6]. Thus, for the dynamic simulation of real life problems, large linear systems have to be solved repeatedly. In this applications, the linear solver often needs about 50 – 80 % of the total amount of computing time [18].

The sparse, unsymmetric, and high dimensional Jacobian matrices, appearing as coefficient matrices of the linear systems, usualy maintain their sparsity structure during the integration over many time steps. For these steps, the Gaussian elimination method can be used with the same ordering of the pivots, in general. Thus, the following approach can be used for the numerical solution of the linear systems in this context. At the beginning of the simulation and at some later time points, e.g. after a discontinuity in the functions have occured, the pivot ordering is determined with the Gaussian elimination method. This step is called *first factorization*. Then a list of pseudo code instructions is generated to perform repeated factorizations of matrices with the same pattern structure, using the same order of pivots. This code contains only the necessary operations for the factorization and thus it works very efficiently. It is defined independently of a computer and can be adapted to exploit the features of vector as well as parallel computers. The factorization of a matrix by using the pseudo code instructions is called *fast second factorization*. This fast second factorization can now be used within the numerical integration process until e.g. a discontinuity in the function appears, the condition number of the matrix becomes too large, or the Newton method does not converge. In the later cases a new first factorization is performed and a new pseudo code is generated. Then the fast second factorization can be used again.

This algorithms are realized in the linear solver GSPAR [12]. By using a new method for the determination of the pivot ordering, the performance of the first factorization can be improved considerably compared to that of GSPAR. This and other methods have recently been implemented in the solver GSPAR2. The solver has been proven successful for the dynamic process simulation of large real life chemical production plants and for the electric circuit simulation as well. Computing times for complete dynamic simulation runs of industrial applications are given. A modification of the solver is used in the new process simulator BOP (*B*lock *O*riented *P*rocess Simulator) [5].

2. The Gaussian elimination method

The Gaussian elimination method

$$PAQ = LU, \tag{2.1}$$

$$Ly = Pb, \qquad UQ^{-1}x = y \tag{2.2}$$

is used for solving the linear system

$$Ax = b. \tag{2.3}$$

The nonzero elements of the matrix A are stored in compressed sparse row format, also known as sparse row wise format. L is a lower triangular 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 GSPAR we consider the following approach for the determination of the matrices P and Q. At each elimination step, the algorithm searches for the first column with a minimal number of nonzero elements. This column becomes the pivot column. Then the columns are reordered. The method can be implemented easily, but the order of the numerical complexity is $O(n^2)$. For keeping the method numerically stable, the pivot $a_{i,j}$ of each elimination step is selected among those candidates satisfying the numerical threshold criterion

$$|a_{i,j}| \geq \beta \max_{l \geq k} |a_{l,j}|,$$

with a given threshold parameter $\beta \in (0, 1]$. This process is called partial pivoting. In our applications we usually choose $\beta = 0.01$ or $\beta = 0.001$.

In GSPAR2 we consider another approach. Here, we first reorder the columns according to the number of nonzeros in ascending order. Therefore, the first column becomes the pivot column of the first elimination step. At the end of each elimination step, the number of nonzeros in the remaining columns is corrected. The algorithm uses linked list techniques and is more complicated as the algorithm for GSPAR. The advantage of the algorithm of GSPAR2 over the algorithm of GSPAR is its minor numerical complexity. The complexity is here only $O(n)$. For the numerical stability, partial pivoting is used as described above.

Matrix	discipline	n	$\|A\|$
bayer01	chemical	57 735	277 774
bayer02	engineering	13 935	63 679
bayer03		6 747	56 196
bayer04		20 545	159 082
bayer06		3 008	27 576
bayer10		13 436	94 926
advice3388	circuit	3 388	40 545
meg1	simulation	2 904	58 142
meg4		5 860	46 842
rlxADC_tr		5 355	32 251
zy3315		3 315	15 985
poli	account of	4 008	8 188
poli_large	capital links	15 575	33 074

TABLE 1. Test matrices

In Table 2 on page 4 the performance of the first factorization with GSPAR2 is compared to that of GSPAR for some test matrices characterized in Table 1. The given CPU times are for a DEC AlphaServer with a processor 21164A operating at 400 MHZ. The number of columns and rows is denoted by n and $|A|$ identifies the number of nonzero elements in the matrix. Many of test matrices can be found in [7].

Matrix	GSPAR	GSPAR2
bayer01	34.92	2.35
bayer02	2.20	0.55
bayer03	0.67	0.30
bayer04	5.18	1.82
bayer06	0.82	0.83
bayer10	3.07	1.27

Table 2. First factorization with GSPAR and GSPAR2, CPU times in seconds

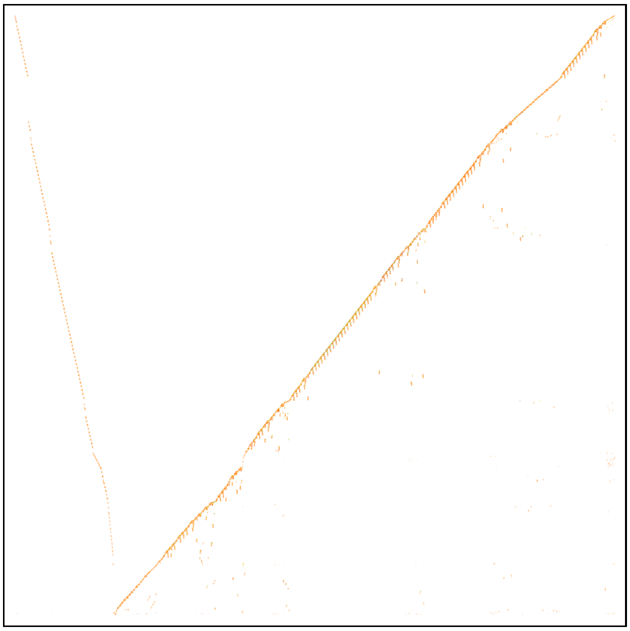


Figure 1. Matrix structure for bayer04

In Figure 1 the structure of the matrix bayer04 is shown. For the creation of the figure, a tool from T. Davis [7] has been used. The matrix bayer04 as well as the other bayer matrices in Table 1 are very unstructured and unsymmetric.

3. Fast second factorization

If initial value problems for large systems of differential–algebraic equations have to be treated numerically, e.g. in the dynamic simulation of large scale applications in electric circuit analysis or chemical process engineering, numerous linear systems have to be solved. In this cases it mostly happens, that the Gaussian elimination method has to be used repeatedly for the factorization of a number of matrices with the same pattern structure. To exploit this circumstance for an efficient implementation, a list of pseudo code instructions for the computation of the factorization is generated.

For the generation of these pseudo code instructions, the factorization of the Gaussian elimination method is used as shown in Figure 2. The algorithm in this form needs exactly n divisions. Only a few different types of pseudo code instructions are needed in this case. The types of the pseudo code instructions and the corresponding lists of indices are stored in integer arrays. For the factorization of matrices with a very large number of nonzero elements, it can happen that the number of integer array elements needed for the storage of the pseudo code instructions exceed main memory limits [12]. In this cases the pseudo code technique can not be used and may be replaced by using FORTRAN subroutines.

```
for $i = 2, n$ do
    $a_{i-1,i-1} = 1/a_{i-1,i-1}$
    for $j = i, n$ do
        $a_{j,i-1} = (a_{j,i-1} - \sum_{k=1}^{i-2} a_{j,k} a_{k,i-1}) a_{i-1,i-1}$
    enddo
    for $j = i, n$ do
        $a_{i,j} = a_{i,j} - \sum_{k=1}^{i-1} a_{i,k} a_{k,j}$
    enddo
enddo
$a_{n,n} = 1/a_{n,n}.$
```

FIGURE 2. Gaussian elimination method

4. Vectorization and Parallelization

The pseudo code instructions can be used for both vectorization as well as parallelization. In both cases, elements of the factorized matrix have to be found which can be computed independently of each other.

For the factorization, a matrix

$$M = (m_{i,j}), \quad m_{i,j} \in \mathbb{N} \cup \{0, 1, 2, \ldots, n^2\}$$

is assigned to the matrix

$$LU = PAQ,$$

where $m_{i,j}$ denotes the level of independence.

The elements with the assigned level zero do not need any operations. All elements with the same level in the factorized matrix (2.1) can be computed independently. First all elements with level one are computed, then all elements with level two and so on. The levels of independence for the matrix elements in (2.1) can be computed with the algorithm of Yamamoto and Takahashi [17]. The algorithm for the determination of the levels of independence $m_{i,j}$ is shown in Figure 3.

For an application on a vector computer, we have to find vector instructions at the different levels of independence [12]. For parallelization, it needs to be distinguished between parallel computers with shared memory and with distributed memory. In the case of parallel computers with shared memory and p processors,

```
M = 0
for  $i = 1, n-1$ do
    for all $\{j : a_{j,i} \neq 0 \ \& \ j > i\}$
        $m_{j,i} = 1 + \max(m_{j,i}, m_{i,i})$
        for all $\{k : a_{i,k} \neq 0 \ \& \ k > i\}$
            $m_{j,k} = 1 + \max(m_{j,k}, m_{j,i}, m_{i,k})$
        enddo
    enddo
enddo
```

FIGURE 3. Algorithm of Yamamoto and Takahashi

we assign the pseudo code for each level of independence in p parts of approximately same size to 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. Then the execution of the next level can be started. If the processors are vector processors then this property can be used analogously. The parallel computer Compaq AlphaServer GS80 6/731 and the IBM 7040-681, pSeries 690 are examples for such computers.

In the case of parallel computers with distributed memory and q processors, the pseudo code for each level of independence is again partitioned into q parts of approximately same size. But in this case, the parts of the pseudo code are moved to the memory of each individual processor. The transfer of parts of the code to the memories of the individual processors is done only once. A synchronization is carried out analogously to the shared memory case. The partitioning and the storage of the matrix as well as of the vectors is implemented in the following way. For small problems the elements of the matrix, right hand side, and solution vector are located in the memory of one processor, whereas for large problems, they have to be distributed over the memories of several processors. We assume that the data communication between the processors for the exchange of data concerning elements of the matrix, right hand side, and solution vector is supported by the operating system. The massive parallel computer Cray T3E is an example for such a computer.

This approach can be illustrated by small artificial example. We assume that the permutation matrices P and Q are given, so that

$$PAQ = \begin{pmatrix} 2 & 4 & & & \\ 5 & 7 & & & 9 \\ & 2 & 9 & & 1 \\ & & 1 & 7 & 8 \\ & & 1 & 3 & 5 \end{pmatrix}. \tag{4.1}$$

The nonzero elements of the matrix A are stored in sparse row format in the vector a. Let $\boxed{\text{i}}$ denote the index of the i-th element in the vector a, then the elements of the matrix PAQ are stored in the following way

$$\begin{pmatrix} \boxed{7} & \boxed{8} & & & \\ \boxed{12} & \boxed{13} & & & \boxed{14} \\ & \boxed{2} & \boxed{1} & & \boxed{3} \\ & & \boxed{9} & \boxed{10} & \boxed{11} \\ & & \boxed{4} & \boxed{5} & \boxed{6} \end{pmatrix}. \tag{4.2}$$

The matrix M assigned to the matrix PAQ is found to be

$$M = \begin{pmatrix} 0 & 0 & & & \\ 1 & 2 & & & 0 \\ & 3 & 0 & & 4 \\ & & 1 & 0 & 5 \\ & & 1 & 1 & 6 \end{pmatrix}. \tag{4.3}$$

We can see from (4.3), that six independent levels exist for the factorization. The detailed instructions for the factorization of the matrix A resulting from (4.1) – (4.3) are shown in Table 3 on page 8.

Let us now consider, for example, the instructions of level one in Table 3 only. Then, at one hand on a vector computer, one vector instruction of the length four can be generated. And at the other hand on a parallel computer with distributed or shared memory, a allocation of the four instructions to two or four processors can be done [12].

From our experiments with many different matrices arising from the process simulation of chemical plants and the circuit simulation respectively, it was found that the number of levels of independence is in general small. The number of instructions in the first two levels is very large, in the next four to six levels it is large and finally it becomes smaller and smaller.

Level	Instructions		
1	a(12)	=	a(12)/a(7)
	a(9)	=	a(9)/a(1)
	a(4)	=	a(4)/a(1)
	a(5)	=	a(5)/a(10)
2	a(13)	=	a(13) − a(12) ⋆ a(8)
3	a(2)	=	a(2)/a(13)
4	a(3)	=	a(3) − a(2) ⋆ a(14)
5	a(11)	=	a(11) − a(5) ⋆ a(3)
6	a(6)	=	a(6) − a(4) ⋆ a(3) − a(5) ⋆ a(11)

TABLE 3. Instructions for the factorization

5. Numerical results

The developed numerical methods are realized in the program packages GSPAR and GSPAR2. The latest version GSPAR2 has been released in 2002. Whereas GSPAR is written in Fortran 77 and implemented on workstations (Digital Alpha-Station, IBM RS/6000, SGI, Sun UltraSparc 1 and 2), vector computers (Cray J90, C90), parallel computers with shared memory (Cray J90, C90, SGI Origin2000, Digital AlphaServer) and parallel computers with distributed memory (Cray T3D), its successor GSPAR2 is written in Fortran 90 and currently implemented on Windows PC's, workstations (Compaq Workstation, SGI, IBM) and parallel computers (Compaq AlphaServer GS80 6/731, IBM 7040-681, pSeries 690).

Numerical results will be given for GSPAR, GSAPAR2, and some other lately released linear solvers. GSPAR has been applied to the test matrices from Table 1 on page 3. In Table 4 on page 9 results are shown using the solver GSPAR on a DEC AlphaServer with an alpha EV5.6 (21164A) processor. Here *# op LU* is the number of operations (only multiplications and divisions) and *fill-in* is the number of fill-ins during the factorization. The CPU time (in seconds) for the first factorization, presented in *First factor.*, includes the times for the analysis as well as for the numerical factorization. The CPU time for the generation of the pseudo code is given in *Pseudo code*.

In Table 5 on page 9, CPU times (in seconds) for the second factorization are shown for the linear solvers UMFPACK V1.0 [9], SuperLU V1.0 with minimum degree ordering of A^TA (upper index *) or of $A^T + A$ (upper index$^+$) [16, 10], and GSPAR, using a DEC AlphaStation with an alpha EV4.5 (21064) processor.

In many applications, mainly in the numerical simulation of physical and chemical problems, the analysis step including ordering and first factorization is performed only a few times, but the second factorization is performed very often.

Therefor the CPU time for the second factorization is essential for the overall simulation time.

Matrix	# op LU	fill-in	First factor.	Pseudo code
bayer01	10 032 621	643 898	35.18	12.72
bayer02	2 095 207	134 546	2.28	1.30
bayer03	1 000 325	64 130	0.68	0.47
bayer04	5 954 718	268 006	5.33	3.93
bayer05	119 740	11 024	0.15	0.03
bayer06	3 042 620	73 773	0.85	1.00
bayer09	364 731	23 145	0.18	0.15
bayer10	5 992 500	227 675	3.05	2.55
advice3388	310 348	9 297	0.38	0.65
meg1	796 797	40 436	0.32	0.40
meg4	420 799	38 784	0.68	0.62
rlxADC_dc	73 612	5 404	0.38	0.13
rlxADC_tr	988 759	47 366	0.85	1.13
zy3315	47 326	8 218	0.12	0.03
poli	4 620	206	0.15	0
poli_large	43 310	10 318	2.38	0.25

Table 4. GSPAR first factorization and generation pseudo code

Matrix	UMFPACK V1.0	SuperLU V1.0	GSPAR
bayer01	5.02	6.70 *	3.20
bayer02	1.13	1.47 *	0.55
bayer03	0.72	0.70 *	0.27
bayer04	3.37	2.77 *	1.70
bayer05	0.13	0.75 *	0.05
bayer06	0.83	0.90 *	0.82
bayer09	0.23	0.23 *	0.10
bayer10	1.60	1.57 *	1.65
advice3388	0.25	0.28 +	0.10
meg1	0.58	1.43 +	0.22
meg4	0.37	0.75 +	0.13
rlxADC_dc	0.15	0.18 +	0.03
rlxADC_tr	0.40	0.90 +	0.30
zy3315	0.15	0.18 +	0.02
poli	0.03	0.07 +	0.00
poli_large	0.13	0.27 +	0.03

Table 5. CPU times for second factorization

GSPAR achieves a good performance for second factorization for all linear systems in Table 5. For systems with a large number of equations, GSPAR is at least two times faster then UMFPACK V1.0 and SuperLU V1.0 respectively.

In Table 6, wall–clock times (in seconds) are shown for the second factorization, using GSPAR with different pivoting on a DEC AlphaServer with four alpha EV5.6 (21164A) processors. The parallelization technique is based on OpenMP [15]. The wall–clock times have been determined with the system routine *gettimeofday*.

processors	bayer01	bayer04	bayer01	bayer04
1	0.71	0.39	1.08	0.43
2	0.54	0.27	0.75	0.29
3	0.45	0.23	0.63	0.25
4	0.49	0.24	0.70	0.30

TABLE 6. Wall–clock times for second factorization

In the following, we compare the performance of some of the new software packages for solving linear systems with sparse matrices on a single CPU of a Compaq AlphaServer GS80 6/731 with alpha EV6.7 (21264A) processors, operating at 731 MHz and on an IBM 7040-681, pSeries 690 with 64–bit POWER4 processors, operating at 1.3 GHz. The used packages are now GSPAR2, UMFPACK V3.0, and WSMP Version 1.7 (Watson Sparse Matrix Packages) [13]. We compare GSPAR2 and UMFPACK V3.0 on the Compaq and GSPAR2 and WSMP Version 1.7 on the IBM for the test matrices in Table 7 respectively.

Matrix	discipline	n	|A|	nnc
bayer01	Process simul.	57 735	277 774	33
big	Device simul.	13 209	91 465	12
circuit_3	Circuit simul.	12 127	48 137	5 682
bayer02	Process simul.	13 935	63 679	28
lhr34c	Process simul.	35 152	764 014	36
meg4	Circuit simul.	5 860	46 842	1 194
bayer04	Process simul.	20 545	159 082	43
nopoly	Device simul.	10 774	70 842	11
circuit_4	Circuit simul.	80 209	307 604	8 900
pesa	Device simul.	11 738	79 566	10
shermanACb	Circuit simul.	18 510	145 149	10 405
bayer10	Process simul.	13 436	94 926	32

TABLE 7. Test matrices

Here, |A| identifies the number of nonzero elements in the matrix and nnc the maximum number of nonzeros in columns.

Matrix	First Fac.	Gen. Code	Sec. Fac.	Solv.	Fast Fac.	NNZ–LU
bayer01	1.583	1.248	1.134	0.094	0.328	932 190
big	3.371	2.658	2.656	0.076	0.727	685 441
circuit_3	0.126	0.013	0.014	0.004	0.005	64 608
bayer02	0.243	0.182	0.155	0.015	0.049	192 752
lhr34c	11.713	12.255	11.579	0.252	2.735	3 065 587
meg4	0.106	0.050	0.042	0.003	0.011	84 694
bayer04	0.887	0.799	0.695	0.038	0.181	458 086
nopoly	0.707	0.516	0.409	0.028	0.126	321 350
circuit_4	2.736	1.005	1.155	0.057	0.098	420 875
pesa	1.326	0.997	0.797	0.043	0.263	476 560
shermanACb	6.846	3.509	3.136	0.053	2.011	544 953
bayer10	0.473	0.394	0.305	0.022	0.102	306 865

Table 8. GSPAR2 on the Compaq GS80 6/731

In Table 8 the results are shown for GSPAR2 on the Compaq GS80 6/731. The CPU times (in seconds) are given for the first factorization in *First Fac.*, for the generation of the pseudo codes in *Gen. Code*, for the second factorization using a Fortran routine in *Sec. fac.*, for the solving in *Solv.* and for the fast second factorization in *Fast Fac.*. The number of nonzero elements in the L– and U–matrix is denoted by *NNZ–LU*.

Matrix	Symb Fac.	Num. Fac.	Solv.	Red. Fac.	NNZ–LU
bayer01	0.583	1.017	0.050	0.917	1 080 365
big	0.117	0.367	0.017	0.300	828 480
circuit_3	0.083	0.450	0.000	0.400	106 286
bayer02	0.117	0.200	0.000	0.167	224 979
lhr34c	1.200	1.933	0.083	1.833	2 975 184
meg4	0.117	0.383	0.000	0.350	55 215
bayer04	0.267	0.383	0.017	0.367	362 366
nopoly	0.083	0.167	0.000	0.150	368 545
circuit_4	15.266	934.746	1.217	930.063	43 569 250
pesa	0.100	0.233	0.000	0.200	491 854
shermanACb	0.833	24.666	0.050	23.949	1 266 626
bayer10	0.133	0.233	0.017	0.200	245 986

Table 9. UMFPACK V3.0 on Compaq GS80 6/731

In Table 9 on page 11 results are given for UMFPACK V3.0 on the GS80 6/731. The solver UMFPACK V3.0 is written in ANSI/ISO C and contains different new algorithms [8]. The CPU times (in seconds) are identified for the symbolic factorization in *Symb. Fac.*, for the numeric factorization in *Num. Fac.*, for solving in *Solv.* and for the redo numeric factorization in *Red. Fac.*

Matrix	First Fac.	Gen. Code	Sec. Fac.	Solv.	Fast Fac.
bayer01	0.840	0.740	0.450	0.020	0.160
big	1.850	1.320	0.820	0.020	0.360
circuit_3	0.080	0.010	0.010	0.010	0.010
bayer02	0.160	0.130	0.070	0.010	0.020
lhr34c	6.260	6.540	3.830	0.070	1.460
meg4	0.070	0.030	0.020	0.000	0.000
bayer04	0.510	0.460	0.250	0.010	0.100
nopoly	0.390	0.270	0.140	0.010	0.050
circuit_4	1.430	0.570	0.480	0.020	0.050
pesa	0.730	0.590	0.290	0.010	0.140
shermanACb	3.020	1.900	0.970	0.010	0.730
bayer10	0.290	0.230	0.120	0.000	0.050

TABLE 10. GSPAR2 on IBM pSeries 690

CPU times (in seconds) are shown for GSPAR2 on an IBM pSeries 690 in Table 10. The description of the headline is the same as in Table 8.

Matrix	Analysis	Factorization	Back substitution	NNZ–LU
bayer01	2.640	0.460	0.180	1 562 405
big	0.290	0.090	0.020	701 524
circuit_3	0.130	0.030	0.010	91 060
bayer02	0.600	0.100	0.040	376 018
lhr34c	2.710	0.470	0.110	2 955 105
meg4	0.080	0.030	0.010	111 676
bayer04	0.860	0.120	0.040	536 935
nopoly	0.190	0.060	0.020	390 502
circuit_4	1.980	0.300	0.120	603 556
pesa	0.210	0.080	0.020	509 346
shermanACb	0.460	0.110	0.040	430 300
bayer10	0.680	0.100	0.030	455 494

TABLE 11. WSMP Version 1.7 on an IBM pSeries 690

In Table 11 on page 12 results are given for the linear solver WSMP Version 1.7 [14] on an IBM pSeries 690. The CPU times (in seconds) are given for the analysis step in *Analysis*, for the factorization step in *Factorization* and for the back substitution step in *Back substitution*.

From the results in the Tables 8 – 11 one can see that the second factorization with GSPAR2 is fast. The symbolic factorization with UMFPACK V3.0 is fast too and the time difference between the factorization and the redo factorization is marginal. The package WSMP has a fast factorization step also for large matrices.

6. Application in process engineering

The more and more integrated modeling in process engineering leads to large scale problems in static and dynamic process simulation. The complex real world process models used in this field usually depend on numerous parameters and are in general highly nonlinear. Using concentrated physical models, high-dimensional systems of nonlinear and differential–algebraic equations have to be solved in static and dynamic process simulation respectively. For that purpose robust and efficient numerical simulation tools are needed. They are urgently necessary to improve process design, analysis, as well as operation in todays process industries.

The hierarchical modular simulation concept developed at WIAS [3, 4, 5] is based on divide and conquer techniques and exploits the modular structure of the process, which in most cases is defined by the hierarchical unit structure of the underlying plant. With it the corresponding system of equations is structured into subsystems according to the units and can be partitioned into m blocks

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

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

which can then be treated almost concurrently within appropriately modified numerical methods. The approach has been implemented in the Block Oriented Process simulator *WIAS-BOP* that uses an own compiler to generate a hierarchically structured data interface from a process description with its modeling language MLPE (Modeling Language for Process Engineering). The numerical solution within *BOP* require a repeated solution of linear systems with the same pattern structure of sparse, unsymmetric coefficient matrices, and with multiple right-hand sides. A slightly modified version of the direct solver GSPAR is used within *BOP* to solve these linear systems. For several real life problems, GSPAR has proven to be a robust and reliable linear solver for this application on parallel computers Cray J90 and SGI Origin 2000. For some distillation processes of Bayer AG Leverkusen, resulting into DAE system with several ten thousand equations, the dynamic simulation with *BOP* has achieved speedup factors of up to 10 in wall-clock time on a Cray J90 with 24 processors. For detailed results in this case, we refer to [5].

Another application of GSPAR in process simulation has been performed using the commercial process simulator SPEEDUP [1]. In SPEEDUP the vector versions of the linear solvers FAMP and GSPAR have been used alternatively. FAMP is the default linear solver in SPEEDUP and optimized on the Cray computer architecture. Table 12 shows two large scale industrial problems of the Bayer AG Leverkusen. The number of differential–algebraic equations are given.

name	chemical plants	equations
bayer04	nitration plant	3 268
bayer10	distillation column	13 436

Table 12. Large scale industrial problems

The problems have been solved on a vector computer Cray C90. The CPU times (in seconds) for complete dynamic simulation runs are shown in Table 13.

name	FAMP	GSPAR	in %
bayer04	451.7	283.7	62.8
bayer10	380.9	254.7	66.9

Table 13. CPU time for complete dynamic simulation

References

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

[2] J. Borchardt, F. Grund, D. Horn, M. Uhle, *MAGNUS – Mehrstufige Analyse großer Netzwerke und Systeme.* Weierstraß–Institut für Angewandte Analysis und Stochastik, Report No. 9, Berlin, 1994.

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

[4] J. Borchardt, K. Ehrhardt, F. Grund, D. Horn, *Parallel Modular Dynamic Process Simulation.* In F. Keil, W. Mackens, H. Voss, J. Werther, editors, *Scientific Computing in Chemical Engineering II*, volume 2, pages 152-159, Springer-Verlag Berlin Heidelberg New York, 1999.

[5] J. Borchardt, *Newton-Type Decomposition Methods in Large-Scale Dynamic Process Simulation.* Comput. Chem. Engng. **25** (2001), 951–961.

[6] K. E. Brenann, S. L. Campbell, L. R. Petzold, *Numerical Solution of Initial–Value Problems in Differential–Algebraic Equations.* North–Holland, New York, 1996.

[7] T. A. Davis, *University of Florida Sparse Matrix Collection.* http://www.cise.ufl.edu/~davis/sparse

[8] T. A. Davis, *UMFPACK V3.0.* http://www.cise.ufl.edu/research/sparse/umfpack/

[9] T. A. Davis, I. S. Duff, *An unsymmetric–pattern multifrontal method for sparse LU factorization.* Tech. Report **TR–94–038**, CIS Dept., Univ. of Florida, Gainsville, FL, 1994.

[10] J. W. Demmel, S. C. Eisenstat, J. R. Gilbert, X. S. Li and J. W. H. Liu, *A Supernodal Approach to Sparse Partial Pivoting.* SIAM J. Matrix Anal. Appl. **20** (1999), 720–755.

[11] C. W. Gear, *Numerical Initial Value Problems in Ordinary Differential Equations.* Prentice–Hall, Inc., Englewood Cliffs, New Jersey, 1971.

[12] F. Grund, *Direct Linear Solvers for Vector and Parallel Computers.* In José M. L. M. Palma and Jack Dongarra and Vicente Hernández, editors, *Vector and Parallel Processing - VECPAR'98*, Lecture Notes in Computer Science **1573**, Springer–Verlag Berlin Heidelberg New York, 1999.

[13] A. Gupta, *Recent Advances in Direct Methods for Solving Unsymmetric Sparse Systems of Linear Equations.* ACM Trans. Math. Softw. **28** (2002), 301–324.

[14] A. Gupta, *WSMP: Watson Sparse Matrix Package.* http://www-users.cs.umn.edu/~agupta/wsmp.html

[15] *OpenMP.* http://www.openmp.org.

[16] X. S. Li, *Sparse Gaussian elimination on high performance computers.* Technical Reports **UCB//CSD-96-919**, Computer Science Division, U.C. Berkeley, Ph.D. dissertation, 1996.

[17] F. Yamamoto, S. Takahashi, *Vectorized LU decomposition algorithms for large–scale nonlinear circuits in the time domain.* IEEE Trans. on Computer–Aided Design **CAD–4** (1985), 232–239.

[18] S. E. Zitney, L. Brüll, L. Lang, R. Zeller, *Plantwide dynamic simulation on supercomputers: modeling a Bayer distillation process.* AIChE Symposium Series **91** (1995), 313–316.

Acknowledgment

The author thanks his coworkers Jürgen Borchardt and Dietmar Horn for useful discussions. The valuable assistance and the technical support from the Bayer AG Leverkusen, the Cray Research Munich and Aspen Technology, Inc., Cambridge, MA, USA are gratefully acknowledged.

Weierstrass Institute for Applied Analysis and Stochastics, Mohrenstrasse 39, 10117 Berlin, Germany
E-mail address: `grund@wias-berlin.de`