Weierstraß–Institut
für Angewandte Analysis und Stochastik

im Forschungsverbund Berlin e.V.

Preprint ISSN 0946 – 8633

Direct Linear Solver for Vector and Parallel Computers

Friedrich Grund

Preprint No. 415

Berlin 1998

Direct Linear Solvers for Vector and Parallel Computers*

Friedrich Grund

Weierstrass Institute for Applied Analysis and Stochastics
Mohrenstrasse 39, 10117 Berlin, Germany
grund@wias-berlin.de
http://www.wias-berlin.de/~grund

Abstract. We consider direct methods for the numerical solution of linear systems with unsymmetric sparse matrices. Different strategies for the determination of the pivots are studied. For solving several linear systems with the same pattern structure we generate a pseudo code, that can be interpreted repeatedly to compute the solutions of these systems. The pseudo code can be advantageously adapted to vector and parallel computers. For that we have to find out the instructions of the pseudo code which are independent of each other. Based on this information, one can determine vector instructions for the pseudo code operations (vectorization) or spread the operations among different processors (parallelization). The methods are successfully used on vector and parallel computers for the circuit simulation of VLSI circuits as well as for the dynamic process simulation of complex chemical production plants.

1 Introduction

For solving systems of linear equations

$$Ax = b, \quad A \in \mathbb{R}^{n\times n}, \quad x, b \in \mathbb{R}^n \tag{1}$$

with non singular, unsymmetric and sparse matrices A, we use the Gaussian elimination method. Only the nonzero elements of the matrices are stored for computation. In general, we need to establish a suitable control for the numerical stability and for the fill-in of the Gaussian elimination method.

For the time domain simulation in many industrial applications structural properties are used for a modular modeling. Thus electronic circuits usually consist of identical subcircuits as inverter chains or adders. Analogously, complex chemical plants consist of process units as pumps, reboilers or trays of distillation columns. A mathematical model is assigned to each subcircuit or unit and they are coupled. This approach leads to initial value problems for large systems of differential–algebraic equations. For solving such problems we use backward differentiation formulas and the resulting systems of nonlinear equations are

* This work was supported by the Federal Ministry of Education, Science, Research and Technology, Bonn, Germany under grants GA7FVB-3.0M370 and GR7FV1.

solved with Newton methods. The Jacobi matrices are sparse and maintain their sparsity structure during the integration over many time steps. In general, the Gaussian elimination method can be used with the same ordering of the pivots for these steps. A pseudo code is generated to perform the factorizations of the matrices and the solving of the systems with triangular matrices efficiently. This code contains only the required operations for the factorization and for solving the triangular systems. It is defined independently of a computer and can be adapted to vector and parallel computers.

The solver has been proven successfully 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. For different linear systems with matrices arising from scientific and technical problems the computing times for several linear solvers are compared.

2 The method

The Gaussian elimination method

$$PAQ = LU, \tag{2}$$
$$Ly = Pb, \quad UQ^{-1}x = y \tag{3}$$

is used for solving the linear systems (1). 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 the following, we consider two cases for the determination of the matrices P and Q.

In the first case, we determine in each elimination step a permutation in the matrix Q. For this, we search the first column with a minimal number of nonzero elements in the matrix to be eliminated. This column becomes the pivot column [6] and the columns are reordered (dynamic ordering). 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

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

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 the second case, we determine in a first step the permutation matrix Q by minimum degree ordering of A^TA or of $A^T + A$, using the algorithm from SuperLU [9]. Then the columns are reordered and in a separate step the permutation matrix P is determined by using partial pivoting.

3 Pseudo code

As mentioned above, it is possible to use the Gaussian elimination method with the same pivot ordering to solve several linear systems with the same pattern structure of the coefficient matrix. To do this, we generate a pseudo code to perform the factorization of the matrix as well as to solve the triangular systems (forward and back substitution).

For the generation of the pseudo code, the factorization of the Gaussian elimination method is used as shown in Fig. 1.

```
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
$a_{n,n} = 1/a_{n,n}$.
```

Fig. 1. Gaussian elimination method

The algorithm needs n divisions. Six different types of pseudo code instructions are sufficient for the factorization of the matrix, four instructions for the computation of the elements of the upper triangular matrix and two of the lower triangular matrix. For computing the elements of the upper triangular matrix one has to distinguish between the cases that the element is a pivot or not and that it exists or that it is generated by fill-in. For the determination of the elements of the lower triangular matrix one has only to distinguish that the element exists or that it is generated by fill-in.

Let l, with $1 \leq l \leq 6$, denote the type of the pseudo code instruction, n the number of elements of the scalar product and $k, m, i_\kappa, j_\kappa, \kappa = 1, 2, \ldots, n$ the indices of matrix elements. Then, the instruction of the pseudo code to compute an element of the lower triangular matrix

$$a(k) = \left(a(k) - \sum_{\kappa=1}^{n} a(i_\kappa) a(j_\kappa) \right) a(m)$$

is coded in the following form

l	n	i_1	j_1	$\cdots$	i_n	j_n	k	m

.

The integer numbers $l, n, i_\kappa, j_\kappa, k$ and m are stored in integer array elements. For l and n only one array element is used.

The structure of the other pseudo code instructions is analogous.

Let μ denote the number of multiplications and divisions for the factorization of the matrix and ν the number of nonzero elements of the upper and lower triangular matrices. Then one can estimate the number of integer array elements that are necessary to store the pseudo code with

$$\gamma(\mu + \nu).$$

At this $\gamma \approx 2.2$ was found to be sufficient for large systems with more than thousand equations while one has to choose $\gamma \approx 4$ for smaller systems.

4 Vectorization and parallelization

The pseudo code instructions are used for the vectorization and the parallelization as well. For the factorization in (2) and for solving the triangular systems in (3), elements have to be found that can be computed independently of each other.

In the case of 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.

In the case of solving the triangular systems, vectors

$$p = (p_i) \quad \textit{and} \quad q = (q_i), \quad p_i, q_i \in \{0, 1, \ldots, n\}$$

are assigned analogously to the vectors x and y from

$$Ly = Pb \quad \textit{and} \quad UQ^{-1}x = y.$$

Here the levels of independence are denoted by p_i and q_i.

The elements with the assigned level zero do not need any operations. Now, all elements with the same level in the factorized matrix (2) as well as in the vectors x and y from (3) 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) and for the vector elements in (3) can be computed with the algorithm of Yamamoto and Takahashi [11]. The algorithm for the determination of the levels of independence $m_{i,j}$ is shown in Fig. 2. The corresponding algorithm for the determination of the elements of the vectors p and q is analogous to it.

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

Fig. 2. Algorithm of Yamamoto and Takahashi

For a vector computer, we have to find vector instructions at the different levels of independence [2,7]. Let $a(i)$ denote the nonzero elements in LU. The vector instructions, shown in Fig. 3, have been proven to be successful in the case of factorization. The difficulty is that the array elements are addressed indirectly. But adequate vector instructions exist for many vector computers. The Cray vector computers, for example, have explicit calls to gather/scatter routines for the indirect addressing.

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

Fig. 3. Types of vector instructions for factorization

For parallelization, it needs to distinguish between parallel computers with shared memory and with distributed memory.

In the case of parallel computers with shared memory and p processors, we assign the pseudo code for each level of independence in 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 is also used. The moderate parallel computer Cray J90 with a maximum number of 32 processors is an example for such a computer.

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 analogous 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, while 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 computers Cray T3D and T3E are examples for such computers.

Now, we consider a small example to illustrate our approach. For a matrix

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

the determination the permutation matrices P and Q gives

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

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

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

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

$$M = \begin{pmatrix} 0 & 0 & & & \\ 1 & 2 & & & 0 \\ & 3 & 0 & & 4 \\ & & 1 & 0 & 5 \\ & & 1 & 1 & 6 \end{pmatrix}. \tag{7}$$

From (7), we can see, that six independent levels exist for the factorization.

The instructions for the factorization of the matrix A resulting from (4) – (7) are shown in Table 1.

Table 1. Instructions for the factorization

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)

Now, we consider, for example, the instructions of level one in Table 1 only. One vector instruction of the length four can be generated (see Fig.3) on a vector computer.

On a parallel computer with distributed memory and two processors, the allocation of the instructions of level one to the processors is shown in Table 2. The transfer of the instructions to the local memory of the processors is done during the analyse step of the algorithm. The data transfer is carried out by the operating system.

Table 2. Allocation of instructions to processors

	processor one	processor two
computation of	a(12), a(9)	a(4), a(5)
	synchronization	

On a parallel computer with shared memory the approach is analogous. The processors have to be synchronized after the execution of the instructions of each level.

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 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.

5 Numerical results

The developed numerical methods are realized in the program package GSPAR. GSPAR is implemented on workstations (Digital AlphaStation, 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).

The considered systems of linear equations result from real life problems in the dynamic process simulation of chemical plants, in the electric circuit simulation and in the account of capital links (political sciences) [1]. The $n \times n$ matrices A with $|A|$ nonzero elements are described in Table 3.

Table 3. Test matrices

name	discipline	n	$\|A\|$
bayer01	chemical	57 735	277 774
b_dyn	engineering	1 089	4 264
bayer02		13 935	63 679
bayer03		6 747	56 196
bayer04		20 545	159 082
bayer05		3 268	27 836
bayer06		3 008	27 576
bayer09		3 083	21 216
bayer10		13 436	94 926
advice3388	circuit	33 88	40 545
advice3776	simulation	3 776	27 590
cod2655_tr		2 655	24 925
meg1		2 904	58 142
meg4		5 960	46 842
rlxADC_dc		5 355	24 775
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

In Table 4 results for the matrices in Table 3 are shown using the method 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)

[1] Some matrices, which are given in Harwell–Boeing format and interesting details about the matrices, can be found in Tim Davis, University of Florida Sparse Matrix Collection, http://www.cise.ufl.edu/~davis/sparse/

for the first factorization, presented in *strat*, 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 *code*. At the one hand, a dynamic ordering of the columns can be applied during the pivoting. At the other hand, a minimum degree ordering of A^TA (upper index *) or of $A^T + A$ (upper index $^+$) can be used before the partial pivoting.

Table 4. GSPAR first factorization and generation pseudo code

	dynamic ordering				minimum degree ordering			
name	# op LU	fill-in	strat.	code	# op LU	fill-in	strat.	code
bayer01	10 032 621	643 898	35.18	12.72	13 860 173	812 505	5.75	9.95 *
b_dyn	15 902	2 909	0.02	0	21 556	8 231	0.02	0.02 *
bayer02	2 095 207	134 546	2.28	1.30	2 030 130	165 357	1.03	2.20 *
bayer03	1 000 325	64 130	0.68	0.47	625 272	53 991	0.25	0.35 *
bayer04	5 954 718	268 006	5.33	3.93	6 340 579	290 021	1.95	2.77 *
bayer05	119 740	11 024	0.15	0.03	474 273	33 797	0.18	0.17 *
bayer06	3 042 620	73 773	0.85	1.00	5 008 097	129 278	1.42	1.52 *
bayer09	364 731	23 145	0.18	0.15	287 947	22 022	0.12	0.12 *
bayer10	5 992 500	227 675	3.05	2.55	3 953 687	203 633	1.28	1.40 *
advice3388	310 348	9 297	0.38	0.65	396 965	9 818	0.75	0.95 $^+$
advice3776	355 465	25 656	0.35	0.75	382 224	26 074	0.62	0.98 $^+$
cod2655_tr	3 331 105	113 640	0.90	1.00	4 839 771	144 875	1.50	1.40 $^+$
meg1	796 797	40 436	0.32	0.40	1 245 847	59 558	0.48	0.78 $^+$
meg4	420 799	38 784	0.68	0.62	376 324	35 008	0.30	0.48 $^+$
rlxADC_dc	73 612	5 404	0.38	0.13	63 227	2 906	0.08	0.08 $^+$
rlxADC_tr	988 759	47 366	0.85	1.13	1 049 623	48 888	0.72	1.13 $^+$
zy3315	47 326	8 218	0.12	0.03	49 263	8 202	0.03	0.02 $^+$
poli	4 620	206	0.15	0	6 094	41	0.02	0 *
poli_large	43 310	10 318	2.38	0.25	34 115	588	0.08	0.03 $^+$

The results in Table 4 show the following characteristics. For linear systems arising from the process simulation of chemical plants, the analyse step with the minimum degree ordering is in most cases, particularly for large systems, faster then with the dynamic ordering, but the fill-in and the number of operations for the factorization are larger. On the other hand, for systems arising from the circuit simulation the factorization with the dynamic ordering is in most cases faster then the minimum degree ordering. The factorization with the minimum degree ordering of A^TA is favourable for systems arising from chemical process simulation, while using an ordering of $A^T + A$ is recommendable for systems arising from the circuit simulation. The opposite cases of the minimum degree ordering are unfavourable because the number of operations and the number of fill-ins is very large.

In Table 5, cpu times (in seconds) for the second factorization are shown for the linear solvers UMFPACK [4], SuperLU with minimum degree ordering of A^TA (upper index *) or of $A^T + A$ (upper index $^+$) [5], Sparse [8] and GSPAR with dynamical column ordering, 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 often. Therefor the cpu time for the second factorization is essential for the overall simulation time.

Table 5. Cpu times for second factorization

name	UMFPACK	SuperLU	Sparse	GSPAR
bayer01	5.02	6.70 *	7.78	3.20
b_dyn	0.05	0.05 *	0.07	0.00
bayer02	1.13	1.47 *	10.433	0.55
bayer03	0.72	0.70 *	17.467	0.27
bayer04	3.37	2.77 *	187.88	1.70
bayer05	0.13	0.75 *	0.08	0.05
bayer06	0.83	0.90 *	54.33	0.82
bayer09	0.23	0.23 *	3.57	0.10
bayer10	1.60	1.57 *	379.75	1.65
advice3388	0.25	0.28 $^+$	0.15	0.10
advice3776	0.30	0.42 $^+$	0.20	0.10
cod2655_tr	0.30	0.55 $^+$	0.27	0.10
meg1	0.58	1.43 $^+$	13.95	0.22
meg4	0.37	0.75 $^+$	0.25	0.13
rlxADC_dc	0.15	0.18 $^+$	0.04	0.03
rlxADC_tr	0.40	0.90 $^+$	0.72	0.30
zy3315	0.15	0.18 $^+$	0.03	0.02
poli	0.03	0.07 $^+$	0.00	0.00
poli_large	0.13	0.27 $^+$	0.04	0.03

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

The cpu times for solving the triangular matrices are one order of magnitude smaller then the cpu times for the factorization. The proportions between the different solvers are comparable to the results in Table 5.

The vector version of GSPAR has been compared with the frontal method FAMP [12] on a vector computer Cray Y–MP8E using one processor. The used version of FAMP is the routine from the commercial chemical process simulator

SPEEDUP [2] [1]. The cpu times (in seconds) for the second factorization are shown in Table 6.

Table 6. Cpu times for second factorization

name	FAMP	GSPAR
b_dyn	0.034	0.011
bayer09	0.162	0.082
bayer03	0.404	0.221
bayer02	0.683	0.421
bayer10	1.290	0.738
bayer04	2.209	0.983

GSPAR is at least two times faster then FAMP for these examples. The proportions for solving the triangular systems are again the same.

For two large examples the number of levels of independence are given in Table 7, using GSPAR with two different ordering for pivoting. The algorithm for lower triangular systems is called forward substitution and the analogous algorithm for upper triangular systems is called back substitution.

Table 7. Number of levels of independence

example		dynamical ordering	minimum degree ordering
	factorization	3 077	3 688
bayer01	forward sub.	1 357	1 562
	back substit.	1 728	2 476
	factorization	876	820
bayer04	forward sub.	399	338
	back substit.	556	495

In Table 8, 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 [10]. The wall–clock times have been determined with the system routine *gettimeofday*.

In Table 9, the cpu times (in seconds) on a Cray T3D are given for the second factorization, using GSPAR with dynamic ordering for pivoting. The linear

[2] Used under licence 95122131717 for free academic use from Aspen Technology, Cambridge, MA, USA; Release 5.5–5

Table 8. Wall–clock times for second factorization

processors	dynamical ordering		minimum degree ordering	
	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

systems can not be solved with less then four or sixteen processors respectively, because the processors of the T3D have not enough local memory for the storage of the pseudo code in this cases. The speedup factors are set equal to one for four or sixteen processors respectively.

Table 9. Cpu times for second factorization on Cray T3D

example	processors	cpu time	speedup factor
	4	1.59	1.00
	8	0.99	1.60
bayer04	16	0.60	2.65
	32	0.37	4.30
	64	0.24	6.63
	16	2.36	1.00
bayer01	32	1.45	1.63
	64	0.95	2.47

6 Applications

Problems of the dynamic process simulation of chemical plants can be modeled by initial value problems for systems of differential–algebraic equations. The numerical solution of these systems [3] involves the solution of large scale systems of nonlinear equations, which can be solved with modified Newton methods. The Newton corrections are found by solving large unsymmetric sparse systems of linear equations. The overall computing time of the simulation problems is often dominated by the time needed to solve the linear systems. In industrial applications, the solution of sparse linear systems requires often more then 70 % of the total simulation time. Thus a reduction of the linear system solution time usually results into a significant reduction of the overall simulation time [13].

Table 10 shows three large scale industrial problems of the Bayer AG Leverkusen. The number of differential–algebraic equations as well as an estimate for the condition number of the matrices of the linear systems are given. The condition numbers are very large, what is typical for industrial applications in this field.

Table 10. Large scale industrial problems

name	chemical plants	equations	condition numbers
bayer04	nitration plant	3 268	2.95E+26, 1.4E+27
bayer10	distillation column	13 436	1.4E+15
bayer01	five coupled distillation columns	57 735	6.0E+18 6.96E+18

The problems have been solved on a vector computer Cray C90 using the chemical process simulator SPEEDUP [1]. In SPEEDUP the vector versions of the linear solvers FAMP and GSPAR have been used alternatively. The cpu time (in seconds) for complete dynamic simulation runs are shown in Table 11.

Table 11. Cpu time for complete dynamic simulation

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

For the large plant bayer01 benchmark tests have been performed on a dedicated computer Cray J90, using the simulator SPEEDUP with the solvers FAMP and GSPAR alternatively. The results are given in Table 12.

Table 12. Bench mark tests

time	FAMP	GSPAR	in %
cpu time	6 066.4	5 565.8	91.7
wall–clock time	6 697.9	5 797.1	86.5

The simulation of plant bayer01 has been performed also on a vector computer Cray C90 connected with a parallel computer Cray T3D, using SPEEDUP and the parallel version of GSPAR. Here, the linear systems have been solved on the parallel computer while the other parts of the algorithms of SPEEDUP have been performed on the vector computer. GSPAR needs 1 440.5 seconds cpu time on a T3D with 64 used processors. When executed on the Cray C90 only, 2 490 seconds are needed for the total simulation.

Acknowledgments. The author thanks his coworkers J. Borchardt and D. 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.

References

1. AspenTech: SPEEDUP, User Manual, Library Manual. Aspen Technology, Inc., Cambridge, Massachusetts, USA (1995)
2. Borchardt, J., Grund, F., Horn, D.: Parallelized numerical methods for large systems of differential–algebraic equations in industrial applications. Preprint No. 382, WIAS Berlin (1997). Surv. Math. Ind. (to appear)
3. Brenan, K.E., Campbell, S.L., Petzold, L.R.: Numerical solution of initial–value problems in differential–algebraic equations. North–Holland, New York (1997)
4. Davis, T.A., Duff, I.S.: An unsymmetric–pattern multifrontal method for sparse LU factorization. Tech. Report TR–94–038, CIS Dept., Univ. of Florida, Gainsville, FL (1994)
5. Demmel, J.W., Gilbert, J.R. Li, X.S.: SuperLU Users' Guide. Computer Science Division, U.C. Berkeley (1997)
6. Grund, F., Borchardt, J., Horn, D., Michael, T., Sandmann, H.: Differential–algebraic systems in chemical process simulation. In Scientific Computing in Chemical Engineering, F. Keil, W. Mackens, H. Voß, J. Werther, eds., Springer–Verlag Berlin Heidelberg (1996) 68–74
7. Grund, F., Michael, T., Brüll, L., Hubbuch, F., Zeller, R., Borchardt, J., Horn, D., Sandmann, H.: Numerische Lösung großer strukturierter DAE–Systeme der chemischen Prozeßsimulation. In Mathematik Schlüsseltechnologie für die Zukunft, K.-H. Hoffmann, W. Jäger, T. Lohmann, H. Schunk, eds., Springer–Verlag Berlin Heidelberg (1997) 91–103
8. Kundert, K.S., Sangiovanni-Vincentelli, A.: Sparse User's Guide, A Sparse Linear Equation Solver. Dep. of Electr. Engin. and Comp. Sc., U.C. Berkeley (1988)
9. Li, Xiaoye S.: Sparse Gaussian elimination on high performance computers. Technical Reports UCB//CSD-96-919, Computer Science Division, U.C. Berkeley (1996), Ph.D. dissertation
10. OpenMP: A proposed standard API for shared memory programming. White paper, http://www.openmp.org (1997)
11. Yamamoto, F., Takahashi, S.: Vectorized LU decomposition algorithms for large–scale nonlinear circuits in the time domain. IEEE Trans. on Computer–Aided Design **CAD–4** (1985) 232–239
12. Zitney, S.E., Stadtherr, M.A.: Frontal algorithms for equation–based chemical process flowsheeting on vector and parallel computers. Computers chem. Engng. **17** (1993) 319–338
13. Zitney, S.E., Brüll, L., Lang, L., Zeller, R.: Plantwide dynamic simulation on supercomputers: Modelling a Bayer distillation process. AIChE Symp. Ser. **91** (1995) 313–316