Solving Unsymmetric Sparse Systems of Linear Equations with PARDISO

Olaf Schenk[1]* and Klaus Gärtner[2]

[1] Department of Computer Science, University of Basel, Klingelbergstrasse 50, CH-4056 Basel, Switzerland
oschenk@ifi.unibas.ch
http://www.ifi.unibas.ch
[2] Weierstrass Institute for Applied Analysis and Stochastics, Mohrenstr. 39, D-10117 Berlin, Germany
gaertner@wias-berlin.de
http://www.wias-berlin.de

Abstract. Supernode pivoting for unsymmetric matrices coupled with supernode partitioning and asynchronous computation can achieve high gigaflop rates for parallel sparse LU factorization on shared memory parallel computers. The progress in weighted graph matching algorithms helps to extend these concepts further and prepermutation of rows is used to place large matrix entries on the diagonal. Supernode pivoting allows dynamical interchanges of columns and rows during the factorization process. The BLAS-3 level efficiency is retained. An enhanced left–right looking scheduling scheme is uneffected and results in good speedup on SMP machines without increasing the operation count. These algorithms have been integrated into the recent unsymmetric version of the PARDISO solver. Experiments demonstrate that a wide set of unsymmetric linear systems can be solved and high performance is consistently achieved for large sparse unsymmetric matrices from real world applications.

1 Introduction.

The solution of large sparse linear systems is a computational bottleneck in many scientific computing problems. When partial pivoting is required to maintain numerical stability in direct methods for solving nonsymmetric linear systems, it is challenging to develop high performance parallel software because partial pivoting causes the computational task-dependency graph to change during runtime. It has been proposed recently that permuting the rows of the matrix prior to factorization to maximize the magnitude of its diagonal entries can be very effective in reducing the amount of pivoting during factorization [1, 9, 10, 15]. The proposed technique, static pivoting, is an efficient alternative to partial pivoting for parallel sparse Gaussian elimination.

* This work was supported by the Swiss Commission of Technology and Innovation KTI under contract number 5648.1.

P.M.A. Sloot et al. (Eds.): ICCS 2002, LNCS 2330, pp. 355–363, 2002.

This paper addresses the issues of improved scalability and robustness of sparse direct factorization within a supernode pivoting approach used in the PARDISO solver[1]. The original aim of the PARDISO project [17,19,20] was to develop a scalable parallel direct solver for sparse matrices arising in semiconductor device and process simulations [4]. These matrices are in general unsymmetric with a symmetric structure, and partial pivoting was not the primary issue during the project. The underlying data structure of the solver is highly optimized and scalability has been achieved with a left-right looking algorithm on shared memory parallel computers [20]. However, after completing the first version of the solver, the authors realized the potential of static pivoting [15] and the use of prepermutations of rows to place large entries on the diagonal. Therefore, the current version is extended towards the efficient solution of large unsymmetric sparse matrices in a shared-memory computing environment.

The suite of unsymmetric test matrices that are used in the experiments througout this paper is shown in Table 1. All matrices are due to real world applications and are publicly available. The table also contains the dimension, the number of nonzeros, and the related application area. It is impossible to solve these linear systems without any pivoting or preordering in many cases.

2 Algorithmic features.

In this section the algorithms and strategies that are used in the analysis and numerical phase of the computation of the LU factors are described.

Supernode pivoting.

Figure 1 outlines the approach to solve an unsymmetric sparse linear system of equations. According to [15] it is very beneficial to precede the ordering by performing an unsymmetric permutation to place large entries on the diagonal and then to scale the matrix so that the diagonals entries are equal to one. Therefore, in step (1) the diagonal matrices D_r and D_c are chosen in such a way that each row and each column of $D_r A D_c$ have a largest entry equal to 1 in magnitude. The row permutation matrix P_r is chosen to maximize the product of the diagonal entries in $P_r D_r A D_c$ with the MC64 code [10]. In step (2) any symmetric fill-reducing ordering can be computed based on the structure of $A + A^T$, e.g. minimum degree or nested dissection. All experiments reported in this paper with PARDISO were conducted with a nested dissection algorithm [14]. Like other modern sparse factorization packages [2,5,7,8,13,16], PARDISO takes advantage of the supernode technology — adjacent groups of rows and columns with the same structure in the factors L and U are treated as one supernode. An interchange among these rows of a supernode has no effect on the overall fill-in and this is the mechanism for finding a suitable pivot

[1] A prebuilt library of the unsymmetric solver PARDISO will be available for several architectures for research purposes at www.ifi.unibas.ch/∼PARDISO in spring 2002.

Table 1. Unsymmetric test matrices with their order (N), number of nonzeros (NNZ), and the application area of origin.

Number	Matrix	N	NNZ	Application
1	af23560	23560	484256	Fluid dynamics
2	av41092	41092	1683902	Finite element analysis
3	bayer01	57735	277774	Chemistry
4	bbmat	38744	1771722	Fluid dynamics
5	comp2c	16783	578665	Linear programing
6	e40r5000	17281	553956	Fluid dynamics
7	ecl32	51993	380415	Circuit Simulation
8	epb3	84617	463625	Thermodynamics
9	fidap011	16614	1091362	Fluid dynamics
10	fidapm11	22294	623554	Fluid dynamics
11	invextr1	30412	1793881	Fluid dynamcis
12	lhr34c	35152	764014	Chemical engineering
13	mil053	530238	3715330	Structural engineering
14	mixtank	29957	1995041	Fluid dynamics
15	nasarb	54870	2677324	Structural engineering
16	onetone1	36057	341088	Circuit Simulation
17	onetone2	36057	227628	Circuit Simulation
18	pre2	659033	5959282	Circuit Simulation
19	raefsky3	21200	1488768	Fluid dynamics
20	raefsky4	19779	1316789	Fluid dynamics
21	rma10	46835	2374001	Fluid dynamics
22	tib	18510	1451491	Circuit simulation
23	twotone	120750	1224224	Circuit simulation
24	wang3	26064	177168	Semicond. dev. simulation
25	wang4	26068	177196	Semicond. dev. simulation

in PARDISO. However, there is no guarantee that the numerical factorization algorithm would always succeed in finding a suitable pivot within the supernode blocks. When the algorithm reaches a point where it can not factor the supernode based on the predescribed inner supernode pivoting, it uses a static pivoting strategy. The strategy suggests to keep the pivotal sequence chosen in the analysis phase and the magnitude of the potential pivot is tested against a threshold of $\sqrt{\epsilon} \cdot ||A||$, where ϵ is the machine precision and $||A||$ is the norm of A. Therefore, in step (3), any tiny pivots encountered during elimination is set to $\sqrt{\epsilon} \cdot ||A||$ — this trades off some numerical stability for the ability to keep pivots from getting to small. The result is that the factorization is in general not exact and iterative refinement may be needed in step (4). If iterative refinement does not converge, an iterative CGS algorithm [21] with the perturbed factors L and U as a preconditioner is used.

The numerical behavior of this approach is illustrated in Table 2, where the number of steps of iterative refinement required to reduce the componentwise relative backward error Berr = $\max_i \frac{|Ax-b|_i}{(|A|\cdot|x|+|b|)_i}$ [3] to machine precision is

(1) Row/column equilibration $A \leftarrow P_r \cdot D_r \cdot A \cdot D_c$, where D_r and D_c are diagonal matrices and P_r is a row permutation to maximize the magnitude of the diagonal entries.
(2) Find a permutation P_c to preserve sparsity: $A \leftarrow P_c \cdot A \cdot P_c^T$ based on $\hat{A} = A + A^T$.
(3) Level-3 Block BLAS factorization $A = P_s L U Q_s$ with supernode pivoting permutations P_s and Q_s. The growth of diagonal elements is controlled with:
    if $(|l_{ii}| < \sqrt{\epsilon} \cdot ||A||)$ **then**
        set $l_{ii} = \sqrt{\epsilon} \cdot ||A||$
    endif
(4) Solve $Ax = b$ using the block L and U factors, the permutation matrices P_c, P_s, P_r, P_s, and Q_s and iterative refinement.
(5) If iterative refinement fails, try CGS with the perturbed factors L and U as a preconditioner.

Fig. 1. Pseudo-code of the supernode pivoting algorithm for general unsymmetric sparse matrices.

shown and the true error is reported as Err = $\frac{||x_{true}-x||}{||x_{true}||}$ (computed from a constant solution $x_{true} = 1$.) It can be seen from the table that it is possible to solve nearly all unsymmetric matrices with the predescribed algorithm. A '*' behind the matrix name indicates that supernode pivoting is necessary to obtain convergence. For matrix pre2 three iteration of the CGS algorithm are necessary to reduce the error by a factor of 10^4, hence the missed subspace is really small.

Parallel LU algorithm with a two-level scheduling

The details of the dynamic scheduling algorithm are described in [18]. The left–right looking approach is writing synchronization data to supernodes that will be factored in the future (right-looking phase) but is reading all numerical data from supernodes processed in the past. To reduce the number of synchronization events and to introduce a smooth transition from tree level to pipelining parallelism the often used central queue of tasks is split into two: the first queue is used to schedule complete subtrees which have a local root node sufficiently far from the root node of the elimination tree. The supernodes inside the subtrees need not any synchronization and hence update the synchronization data of the supernodes of the second kind only (those close to the root node — which are in general large). These supernodes are scheduled by a second queue — contrary to the first task queue the second one keeps track of individual outer supernode updates. Whenever a process can not continue with processing outer updates due missing factorization results it puts the partially processed supernode back into the second queue of tasks and fetches any other supernode la-

Table 2. The numerical behavior of the PARDISO supernode pivoting approach. Err indicates the error, Berr the backward error, Res the norm of the residual, and Nb the number of steps of iterative refinement. A '*' after the matrix name indicates that supernode pivoting is necessary to obtain convergence. 'CGS' indicates that a CGS iterations is used to improve the solution.

Matrices	Err	Berr	Res	Nb	Err	Berr	Res
af23560	1.4e-12	2.8e-13	5.4e-11	2	2.5b-13	3.2e-16	1.6e-11
av41092	fail	fail	fail	fail	fail	fail	fail
bayer01*	1.0e-05	1.0e-08	2.6e-10	2	1.4e-06	3.6e-16	6.4e-14
bbmat	1.1e-08	3.6e-16	1.0e-11	1	1.2e-08	3.5e-16	8.6e-12
comp2c	2.7e+01	2.7e-04	5.1e-03	4	6.9e-07	2.0e-16	1.0e-12
e40r5000	1.8e+03	1.5e-02	1.6e+00	4	1.9e-10	1.1e-16	3.2e-11
ecl32	1.6e-03	2.5e-08	4.6e-06	3	6.2e-11	2.6e-16	3.8e-12
epb3	6.0e-12	8.9e-14	1.2e-14	2	9.6e-13	2.5e-16	5.1e-15
fidap011*	2.6e-04	2.8e-15	2.3e-06	2	2.6e-06	4.1e-16	1.2e-06
fidapm11*	4.6e-01	1.4e-04	6.0e-06	11	3.4e-12	2.8e-16	4.0e-15
invextr1*	1.4e+04	2.0e-05	5.4e+0	5	1.8e-06	1.1e-15	1.9e-06
mil053	3.7e-10	2.5e-12	8.7e-10	2	9.2e-10	2.3e-16	2.5e-11
mixtank*	8.8e-04	1.7e-14	2.5e-09	4	5.7e-11	3.5e-16	1.9e-14
nasarb*	8.5e-08	8.8e-14	6.5e-06	2	8.5e-10	4.8e-16	6.5e-06
onetone1	4.2e-10	6.6e-13	4.0e-10	2	6.9e-11	3.2e-16	1.3e-11
onetone2	5.1e-10	3.3e-12	2.5e-10	2	6.9e-11	2.3e-16	1.3e-11
pre2	2.3e-01	1.9e-01	2.6e-01	3 CGS	4.3e-05	4.0e-15	3.4e-05
raefsky3	2.9e-09	1.3e-10	5.5e-14	2	3.1e-16	4.6e-16	2.6e-16
raefsky4	8.3e-08	4.4e-09	1.2e-02	2	1.4e-12	4.0e-16	1.6e-04
rma10	1.3e-10	4.4e-16	1.4e-07	2	1.3e-10	4.3e-16	1.43-07
tib	2.2e-09	1.6e-12	5.5e-12	2	6.9e-12	5.5e-16	5.6e-13
twotone	2.2e-08	7.8e-13	1.4e-10	2	1.0e-10	7.8e-16	5.8e-11
wang3	6.4e-10	1.1e-14	3.4e-15	2	1.1e-11	2.5e-16	4.3e-16
wang4	1.0e-11	5.6e-15	4.2e-15	2	1.7e-12	2.6e-16	4.2e-16

beled as executable. This scheme exploits the larger granularity of outer updates (down to single dense matrix-matrix multiply operations), does not force small outer updates to be handled as individual tasks and is open to priority control strategies.

Unfortunately the complexity is already large. The introduction of factorization time dependence is possible, but the additional updates due to out of supernode pivoting have to be handled without a serious degradation of the parallel performance reached by the scheme up to now.

3 Experimental results.

Table 3 list the performance numbers of some state-of-the art packages for solving large sparse systems of linear equations on a single IBM Power 3 processor.

Table 3. LU factorization times (in seconds) on a single 375 Mhz IBM power 3 processor for UMFPACK 3, MUMPS, WSMP, and PARDISO (with prereordering MC64 and METIS) respectively. The best time is shown in boldface, the second best time is underlined, and the best operation count is indicated by □. The last row shows the approximate smallest relative pivot threshold that yielded a residual norm close to machine precision after iterative refinement for each package ([12, 13]).

Matrices	MUMPS		UMFPACK 3		WSMP		PARDISO	
	time (in sec)	ops $\times 10^9$	time (in sec)	ops $\times 10^9$	time (in sec)	ops $\times 10^9$	time (in sec)	ops $\times 10^9$
af23560	**3.89**	2.56	9.07	3.46	3.96	3.27	4.52	3.33
av41092	21.0	10.9	128.	37.4	**4.56**	2.14	fail	fail
bayer01	2.54	.697	1.11	.024	0.95	.040	**0.57**	.126
bbmat	54.3	41.6	88.3	39.1	**22.9**	20.1	52.6	27.9
comp2c	10.5	4.84	597.	113.	**1.64**	0.78	9.20	2.08
e40r5000	14.5	5.43	6.76	2.09	1.08	.521	**0.85**	.456
ecl32	64.2	64.6	191.	112.	**23.1**	21.0	29.9	22.4
epb3	2.84	1.17	5.77	1.34	1.66	.452	**1.54**	.441
fidap011	8.58	7.01	18.5	8.51	**3.93**	3.20	4.51	3.65
fidapm11	11.9	10.0	40.9	20.0	**6.50**	5.21	11.9	8.91
invextr1	80.7	71.5	178.	89.4	**9.93**	6.90	16.7	12.3
mil053	43.5	31.8	107.	46.2	23.0	14.4	**22.9**	13.7
mixtank	151.	141.	398.	243.	**21.9**	19.5	88.5	80.1
nasarb	12.8	9.45	55.9	28.2	**6.98**	5.41	9.42	7.02
onetone1	17.1	8.19	5.58	2.33	**2.25**	1.25	4.14	2.25
onetone2	1.67	.605	.760	.080	**.720**	.191	.901	.438
pre2	fail	fail	fail	fail	**127.**	96.3	fail	fail
raefsky3	4.44	2.90	16.0	7.87	**3.16**	2.57	3.22	2.63
raefsky4	107.	74.4	26.6	12.9	**4.91**	4.11	5.83	4.73
rma10	4.00	1.39	8.83	3.44	**2.47**	1.48	3.20	1.92
tib	.560	.122	28.1	.203	.350	.064	**.201**	.032
twotone	56.5	38.3	31.6	10.8	**13.5**	9.46	59.3	37.4
wang3	72.9	57.8	40.6	24.2	6.65	5.91	**5.88**	4.58
wang4	11.8	10.5	53.4	30.7	6.84	6.09	**3.94**	3.02
Thresh	0.01		0.20		0.01		–	

This table is shown to locate the performance of PARDISO against other wellkown software packages. A detailed comparison can be found in [12, 13]. A "fail" indicates that the solver ran out of memory, e.g. MUMPS [2], UMFPACK [6], or the iterative refinement did not converge, e.g PARDISO. The default option of the PARDISO was a nested dissection ordering and a prepermutation with MC64 for all matrices.

Table 4. Operation count (Ops), LU factorization time in seconds, and speedup (S) of WSMP and PARDISO on one (T_1) and four (T_4) 375 MHz IBM Power 3 processors with default options. The best time with four processors is shown in boldface, the best time with one processor is underlined.

	WSMP				PARDISO			
Matrices	ops $x10^9$	T_1 (s)	T_4 (s)	S $\frac{T_1}{T_4}$	ops $x10^9$	T_1 (s)	T_4 (s)	S $\frac{T_1}{T_4}$
af23560	3.27	3.96	2.27	1.8	3.33	4.52	**1.27**	3.6
bayer01	1.57	0.95	0.95	1.0	.126	.571	**0.31**	1.8
bbmat	20.1	22.9	**8.26**	2.8	27.9	52.6	13.6	3.9
comp2c	0.78	1.64	**0.67**	2.4	2.08	9.20	2.81	3.3
ecl32	21.0	23.1	**7.41**	3.1	22.4	29.9	8.81	3.4
epb3	.452	1.66	1.23	1.4	.441	1.54	**.811**	1.9
fidap011	3.20	3.93	1.78	2.2	3.65	4.51	**1.54**	2.9
invextr1	6.90	9.93	**4.67**	2.1	12.3	16.7	5.23	3.2
lhr34c	.163	0.92	0.93	1.0	.552	2.60	**0.90**	2.9
mil053	14.4	23.0	10.6	2.2	13.7	22.9	**9.91**	2.3
nasarb	5.41	6.98	3.37	2.1	7.02	9.42	**2.49**	3.8
onetone1	1.25	2.25	**1.52**	1.5	2.25	4.14	1.85	2.2
onetone2	.191	0.72	0.72	1.0	.438	.901	**0.31**	2.9
raefsky3	2.57	3.16	1.40	2.3	2.63	3.22	**0.92**	3.5
raefsky4	4.11	4.91	2.34	2.1	4.73	5.83	**1.69**	3.4
rma10	1.48	2.47	**0.99**	2.5	1.92	3.20	1.04	3.1
twotone	9.46	13.5	**9.05**	1.5	37.4	59.3	17.8	3.3
venkat50	1.75	2.83	1.13	2.5	1.83	3.21	**0.93**	3.4
wang3	5.91	6.65	3.50	1.9	4.58	5.88	**1.82**	3.2
wang4	6.09	6.84	3.08	2.2	3.02	3.94	**1.27**	3.1

For the parallel performance and scalability, the LU factorization of PARDISO is compared with that of WSMP in Table 4. The experiments were conducted with one and four IBM 375 Mhz Power 3 processors. The four processors all have a 64 KB level-1 cache and a four MB level-2 cache. WSMP uses the Pthread libray and PARDISO uses the OpenMP parallel directives. Both solver always permute the original matrix to maximize the product of diagonal elements and nested-dissection based fill-orderings has been used [11, 14]. Two important observation can be drawn from the table. The first is that WSMP needs in most of the examples less operations than PARDISO. It seems that the algorithm based on [11] produces orderings with a smaller fill-in compared with [14], which is used in PARDISO. The second observation is that the factorization times are affected by the preprocessing and WSMP is in most cases faster on a single Power 3 processor. However, the two-lewel scheduling in PARDISO provides better scalability and hence better performance with four Power 3 processors.

4 Concluding remarks.

The focus of the comparison is mainly on the WSMP and the PARDISO packages[2] and their different approaches:
— stability with a general pivoting method and the dynamic directed acyclic task dependency graphs in WSMP and
— supernode pivoting with a preordering, and undirected graph partioning in PARDISO, where the efficient dynamic scheduling on precomputed communication graphs results in better speedup.
For different application areas the PARDISO approach results in the needed stability. It has reached a development status that can be improved mainly in the following directions:

1. Better reordering schemes to reduce the operation count.
2. Adding dynamic pivoting on the basis of introducing exceptions (what seems to be a justified assumption for the test matrices used – this set produces only a few necessary pivots outside the supernodes at the very beginning of the factorization process), and
3. Excluding some of the systematic operations with zeros in a postprocessing step without loosing the advantages of the left–right looking supernodal approach.

The different techniques used in both approaches may stimulate further improvements but it may be hard to reach the robustness and the operation counts of WSMP for unsymmetric matricces and the better scalability of PARDISO on SMPs.

Acknowledgments.

The authors wish to thank Anshul Gupta, IBM T.J. Watson Research Center, for providing his large benchmark set of unsymmetric matrices, Iain Duff for the possibility to use the MC64 graph matching code, and the Computing Center at the University of Karlsruhe for supporting access to the IBM NightHawk-II parallel computers.

References

1. P. R. Amestoy, I. S. Duff, J.-Y. L'Excellent, and X. S. Li. Analysis and comparison of two general sparse solvers for distributed memory computers. Technical Report TR/PA/00/90, CERFACS, Toulouse, France, December 2000. Submitted to *ACM Trans. Math. Softw.*
2. Patrick R. Amestoy, Iain S. Duff, Jean-Yves L'Excellent, and Jacko Koster. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM J. Matrix Analysis and Applications*, 23(1):15–41, 2001.

[2] The data presented go directly back to the authors at sufficiently close points in time.

3. Mario Arioli, James W. Demmel, and Iain S. Duff. Solving sparse linear systems with sparse backward error. *SIAM J. Matrix Analysis and Applications*, 10:165–190, 1989.
4. R.E. Bank, D.J. Rose, and W. Fichtner. Numerical methods for semiconductor device simulation. *SIAM Journal on Scientific and Statistical Computing*, 4(3):416–435, 1983.
5. T. A. Davis and I. S. Duff. An unsymmetric-pattern multifrontal method for sparse LU factorization. *SIAM J. Matrix Analysis and Applications*, 18(1):140–158, 1997.
6. Timothy A. Davis. UMFPACK. Software for unsymmetric multifrontal method. In *NA Digest, 01(11), March 18, 2001.*, http://www.cise.ufl.edu/research/sparse/umfpack.
7. J. Demmel, J. Gilbert, and X. Li. An asynchronous parallel supernodal algorithm to sparse partial pivoting. *SIAM Journal on Matrix Analysis and Applications*, 20(4):915–952, 1999.
8. 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 Analysis and Applications*, 20(3):720–755, 1999.
9. I. S. Duff and J. Koster. The design and use of algorithms for permuting large entries to the diagonal of sparse matrices. Technical Report TR/PA/97/45, CERFACS, Toulouse, France, 1997. Also appeared as Report RAL-TR-97-059, Rutherford Appleton Laboratories, Oxfordshire.
10. I. S. Duff and J. Koster. The design and use of algorithms for permuting large entries to the diagonal of sparse matrices. *SIAM J. Matrix Analysis and Applications*, 20(4):889–901, 1999.
11. A. Gupta. Fast and effective algorithms for solving graph partitioning and sparse matrix ordering. *IBM Journal of Research and Development*, 41(1/2):171–183, January/March 1997.
12. A. Gupta. Improved symbolic and numerical factorization algorithms for unsymmetric sparse matrices. Technical Report RC 22137 (99131), IBM T. J. Watson Research Center, Yorktown Heights, NY, August 1, 2001.
13. A. Gupta. Recent advances in direct methods for solving unsymmetric sparse systems of linear equations. Technical Report RC 22039 (98933), IBM T. J. Watson Research Center, Yorktown Heights, NY, April 20, 2001.
14. G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1998.
15. X.S. Li and J.W. Demmel. A scalable sparse direct solver using static pivoting. In *Proceeding of the 9th SIAM conference on Parallel Processing for Scientic Computing*, San Antonio, Texas, March 22-34,1999.
16. E.G. Ng and B.W. Peyton. Block sparse Cholesky algorithms on advanced uniprocessor computers. *SIAM Journal on Scientific Computing*, 14:1034–1056, 1993.
17. O. Schenk. *Scalable Parallel Sparse LU Factorization Methods on Shared Memory Multiprocessors*. PhD thesis, ETH Zürich, 2000.
18. O. Schenk and K. Gärtner. Two-level scheduling in PARDISO: Improved scalability on shared memory multiprocessing systems. Accepted for publication in *Parallel Computing*.
19. O. Schenk and K. Gärtner. PARDISO: a high performance serial and parallel sparse linear solver in semiconductor device simulation. *Future Generation Computer Systems*, 789(1):1–9, 2001.
20. O. Schenk, K. Gärtner, and W. Fichtner. Efficient sparse LU factorization with left-right looking strategy on shared memory multiprocessors. *BIT*, 40(1):158–176, 2000.
21. P. Sonneveld. CGS, a fast Lanczos-type solver for nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 10:36–52, 1989.