Efficient Reordering for Direct Methods in Analog Circuit Simulation

Ingo Naumann , Heinz K. Dirks

***Abstract*— For linear systems of equations arising from circuit simulation, we studied the efficiency of the reordering algorithm integrated in the MA27 subroutine package from Duff and Reid. Our conclusion is that the algorithm performs extraordinarily well, provided that the nearly-dense rows and columns of the matrix have been removed prior to reordering. Thus, we recommend the reordering implementation of Duff and Reid for circuit simulation. The reordering times of typical industrial examples confirm this recommendation.**

***Index Terms*— Circuit simulation, Markowitz algorithm, minimum degree, direct methods, Gaussian elimination**

I. Introduction

FINDING the optimal ordering of circuit equations is an NP-complete problem [15]. Applying the Modified Nodal Analysis (MNA) and backward difference formulae (BDF) the output data of a transient analysis are determined by solving at each time step a system of nonlinear algebraic equations. Each nonlinear system is usually solved using the Newton-Raphson method. Moreover, each iteration step of the Newton algorithm itself needs the solution of a system of linear equations which can be written in the form

$$\underset{\sim}{A}\,\underset{\sim}{x} \;=\; \underset{\sim}{b}. \tag{1}$$

A meaningful interpretation of the output data normally requires a run for some thousand time steps while Newton's method typically needs three to four iterations to yield the solution of each system of nonlinear equations. Thus, the efficient solution of the linear equations plays a critical role for the total computation time. Because most of the network nodes are only connected to three or four other nodes, the matrix $\underset{\sim}{A}$ is typically very sparse, i.e. most of their coefficients are zero. For these reasons we require a specially adapted and very fast implementation of a solver that only operates on the nonzero coefficients (or *entries*). While the numerical values of the coefficients change during the solution process, the *matrix structure*, i.e. the pattern of the nonzero cefficients of the matrix, remains the same during the entire simulation since it only depends on the topological structure of the network. A setup routine recalculates the values for the *original entries* (those which are not fill-ins, see below) of $\underset{\sim}{A}$ for every iteration step of the Newton algorithm. In most parts of the matrix the diagonal elements dominate the coefficients of their corresponding row. As for symmetry, the structure of the matrix is normally nearly symmetric; therefore, a structural symmetrization does not implicate any major disadvantage.

Ingo Naumann obtained his doctoral degree in electrical engineering from the Christian-Albrechts-University at Kiel, Germany, and is now working for the German Federal Office for Information Security (BSI).

Heinz K. Dirks is full professor of electrical engineering at the Faculty of Engineering of the Christian-Albrechts-University at Kiel, Germany.

```
for k = 1 to n
  for i = k+1 to n
    a_ik = a_ik / a_kk
    for j = k+1 to n
      a_ij = a_ij - a_ik a_kj
    b_i = b_i - a_ik b_k
```

(a) Form kij

```
for i = 1 to n
  for k = 1 to i-1
    a_ik = a_ik / a_kk
    for j = k+1 to n
      a_ij = a_ij - a_ik a_kj
    b_i = b_i - a_ik b_k
```

(b) Form ikj

Fig. 1. Direct elimination

In this article we consider the use of a direct solver, for example the *Gaussian elimination* or Form kij [3] named after the sequence of the loop indices i,j and k performing the update operation

$$a_{ij} = a_{ij} - \frac{a_{ik}}{a_{kk}} a_{kj} \tag{2}$$

as shown in Fig. 1(a). We have omitted the back-substitution here. This algorithm normally creates fill-ins, i.e. during the solution process some zero coefficients become nonzeros. An appropriate rearrangement of the rows and columns in the matrix (including the right-hand side and the solution vector) can drastically reduce the number of fill-ins and therefore the number of update operations. Since the matrix structure remains fixed, a single determination of the ordering of the equations is sufficient for the entire simulation. We will refer to this process as the *symbolic phase* since this static pivoting is performed prior to any numerical solution in the subsequent *solution phase*. During the symbolic phase the actual values of the coefficients do not have to be considered. It is of course mandatory to keep track of all reordering steps and to provide an interface for the *setup routine* which alternates with the solver during the solution phase: the setup routine supplies the matrix coefficients for the next iteration step and the solver computes the solution of the linear equations. To get a better understanding of the nesting of the different components of the entire simulation process, see [9].

For many fields of applications, the algorithm described by *Markowitz* in 1957 [10] serves as the standard method for a reordering with a minimum fill-in objective. It determines the pivot elements one after the other by calculating a so-called Markowitz criterion $M(x, y)$ for each nonzero a_{xy} of the remaining (not yet selected) rows and columns of $\underset{\sim}{A}$,

$$M(x,y) \quad = \quad (c_x - 1) \cdot (r_y - 1), \tag{3}$$

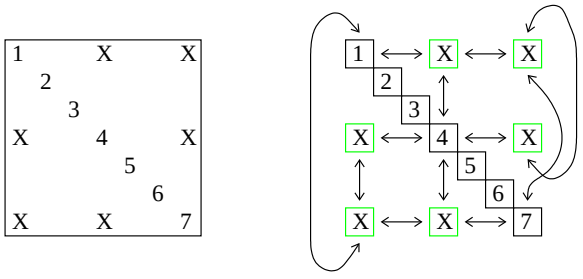
where c_x is the number of entries in column x and r_y is the number of entries in row y. At each step it selects the element with the smallest Markowitz criterion as pivot element. After that, the algorithm determines the fill-ins, removes the row and column corresponding to the pivot element and recalculates the Markowitz criteria for the reduced submatrix.

In circuit simulation, the pivot is normally limited to diagonal elements due to numerical reasons. The fact that the matrix structure is symmetric or nearly symmetric can usually be exploited. We will refer to the Markowitz strategy with diagonal pivoting for a symmetrically structured matrix as the *minimum degree algorithm* [5].

Remember that a circuit simulator has to perfom this reordering just once which is one of the main reasons favouring direct solvers. By applying sophisticated techniques like scatter-gather (this requires the Form ikj) direct solvers achieve an excellent performance [14]. There has been some discussion about the use of certain iterative methods (to solve the systems of linear equations arising in circuit simulation) which are faster than direct methods [2]. The rate of convergence of these methods might however be quite slow in some ill-conditioned cases and an alternative direct solver is integrated in most circuit simulators.

Mostly, it is assumed that solution time always dominates reordering time. According to design tool developers' experiences we want to call this assumption into question and analyze whether and under which conditions it is still valid. In fact, the symbolic phase can certainly turn out as the bottleneck of the simulation and deserves a very careful implementation.

II. Data structures



(a) Matrix structure (b) Data structure

Fig. 2. Possible data structure for the Markowitz algorithm

A first approach of implementing an efficient Markowitz algorithm leads to the data structure as shown in Fig. 2. This approach is straightforward enough to act as a reference for more efficient implementations. During the symbolic phase we only need information about the matrix pattern, therefore no coefficients have to be stored. The essence of this data structure is that an array holds the diagonal elements which are connected to the other entries through pointers. Each data element contains the row and column number of the corresponding entry plus four pointers to other data elements. Two of them reference the adjacent data elements in the same row; the other two are pointers to two arbitrary data elements of the same column. In other words, the elements of each row form a double linked list in column order; the elements of one column form a double linked list in arbitrary order. The reason lies in the way the Markowitz algorithm, or to be more correct, a typical Markowitz *implementation* updates the sparsity pattern of the reduced submatrix: the nonzero entries in the pivotal column give the information which rows might change but the examination of these rows can be performed in any order. On the other hand, the easiest way to determine the fill-in positions of one particular row is to compare its nonzero entries with those of the pivotal row from the left to the right, therefore the row-wise concatenation of one row should be in this order.

To economize in storage we may use the orthogonal linked list [9] where each element is only linked to the right and the lower neighbour but this causes a loss in performance: After selecting the pivot row and column and symbolically inserting the resulting fill-ins, the data elements of the chosen row and column have to be removed from the set of entries. This removal is much faster if the data elements contain pointer to all four directions since this avoids searching through entire rows or columns to reconnect the remaining elements.

Nevertheless, the scientific community offers more efficient implementations of reordering algorithms for direct solvers. Duff and Reid propose in their package MA27 a very sophisticated solution to this problem [4]. Although a naive use of this implementation for circuit simulation might turn out very ineffective in many cases, we can show that it works remarkably well, as long as the matrices comply with certain requirements. We will describe these requirements in the following.

III. Nearly-Dense Rows and Columns

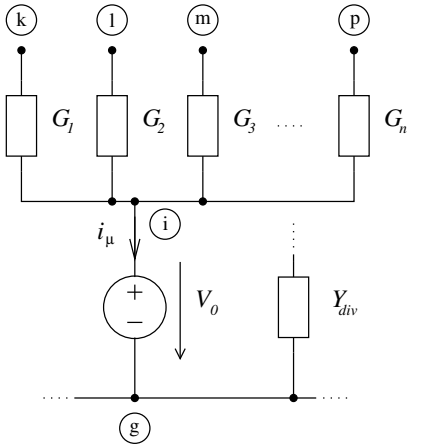


Fig. 3. Example of power supply node

We begin with what we will refer to as *nearly-dense rows and columns*. These are rows and columns with so many

entries that they should not considered as sparse. For instance, the matrix pattern of the example DRA82[1] contains a row with 130370 entries while the matrix dimension is 138679. Roughly, every row or column with more entries than 10% of the matrix dimension falls into our definition of nearly-dense. They occur in all of our examples and they usually constitute less than 0.1% of all rows and columns.

One of the most frequent reasons for a nearly-dense row and a nearly-dense column in the matrix structure is an ideal voltage source node with many connections, as depicted in Fig. 3. This example involves entries in the matrix and the right-hand side of the system of linear equations as shown in the submatrix in Fig. 4. Note that the matrix also contains row and column μ, where row μ corresponds to the branch constituent relation of the ideal voltage source and column μ corresponds to the variable i_μ.

Since the diagonal coefficient $a_{\mu\mu}$ equals zero, a special reordering might be necessary here before a direct elimination method can be applied. We can avoid the zero on the diagonal position simply by swapping the positions of the rows i and μ, as shown in Fig. 5. If in this case g is the ground node, the respective row and column are neglected. Thus, the nearly-dense row corresponds to a column which is empty, except for the diagonal element. Likewise, the nearly-dense column corresponds to a row that contains only its diagonal element. For more details on row interchange caused by ideal voltage sources, see [7].

In the following we describe the special treatment of the few nearly-dense rows and columns *a priori* to the actual reordering phase; it is quite similar to the pre-processing method of the AMD routines from Amestoy, Davis and Duff [1]. Hereafter, we will refer to this as the *removal of nearly-dense rows and columns (from the matrix structure)*. In particular, we distinguish between two cases:

i) We assemble all nearly-dense rows and columns whose corresponding counterparts are also nearly-dense (for example row/column i in Fig. 4) in the lower right corner of the matrix. These rearrangements seem obvious because they avoid producing a large number of fill-ins.

ii) Nearly-dense rows that have only the diagonal entry in the corresponding column (for example row i in Fig. 5) can be moved to the top of the matrix without causing any fill-in. The same applies to equivalent nearly-dense columns. Note that there are less update operations if these rows/columns are in the upper/left part of the matrix than otherwise, even though such a replacement does not affect the number of fill-ins.

The results of these actions coincide with what the Markowitz strategy would normally yield anyway, so keep in mind that the goal to perform these steps beforehand is to save unnecessary calculation time of the reordering routine.

[1]The circuits DRA82, CCP and CIRC2A are three of twenty matrix examples we used for this study. Please note that we cannot publish details on the circuits due to confidentiality restrictions.

	k	l	m	⋯	p	i	g	μ	rhs
k	G_1					$-G_1$			
l		G_2				$-G_2$			
m			G_3			$-G_3$			
⋮				⋱		⋮			
p					G_n	$-G_n$			
i	$-G_1$	$-G_2$	$-G_3$	⋯	$-G_n$	$\sum_\nu G_\nu$		1	
g							Y_{div}	-1	
μ						1	-1	0	V_0

Fig. 4. Stamp of the circuit example (Fig. 3)

	k	l	m	⋯	p	i	g	μ	rhs
k	G_1					$-G_1$			
l		G_2				$-G_2$			
m			G_3			$-G_3$			
⋮				⋱		⋮			
p					G_n	$-G_n$			
μ						1	-1	0	V_0
g							Y_{div}	-1	
i	$-G_1$	$-G_2$	$-G_3$	⋯	$-G_n$	$\sum_\nu G_\nu$		1	

Fig. 5. Stamp after special reordering

IV. The implementation MA27 of Duff and Reid

Duff and Reid provide the scientific community with their sparse matrix package MA27 (part of the Harwell Subroutine Library) which is available in the internet [16]. While the original package is written in FORTRAN we used a C/C++ translation in this study [4], [12] which does not differ from the FORTRAN version in terms of CPU time. The package is designed for symmetrically structured matrices and contains reordering and solving routines for sparse systems of linear equations. The reordering routines are extremely efficient due to successive transactions of so-called clique amalgamations, meaning each new row is created by merging old rows. Therefore, an explicit storage of the fill-in positions is not necessary. A detailed description of this method can be found in [5], [8].

Tab. 1 shows the reordering times of the approach as described in section II and of the implementation MA27 for six circuits that vary widely in their characteristics. For the examples DRA82, CCP and CIRC2A the reordering times for the *reduced* matrix structure after the removal of nearly-dense rows and columns are given in comparison to the original values. Note that the smaller dimension of the reduced system only takes effect during the reordering process; the solver always uses the full-sized system as input. As for

HCIRCUIT and SCIRCUIT, the reduction does not cause any major difference in the computation times.

The results show that the implementation MA27 is usually more efficient than our first approach. Considering the size of the examples, a typical simulation time for a transient simulation of these cricuits would be several hours. Compared to this, reordering times of several seconds can be neglected. On the other hand, the table shows how important the prior removal of the nearly-dense rows and columns turns out to be in general. The removal itself takes less than one second for all examples.

In the last part of this study we discuss some situations where the implementation MA27 is less suited to circuit simulation. If for some reason the matrix structure is strongly unsymmetric, the minumum degree method cannot be used for reordering. Duff and Reid provide an implementation for the unsymmetric case as well, the code MA37. Since the symmetric version of the Markowitz strategy in general runs significantly faster [5] than the unsymmetric one and strongly unsymmetric matrix structures rarely appear[2] in circuit simulation we did not investigate further in this direction.

As is the case in other scientific fields, levelized incomplete LU factorization known as LILU or ILU may be used in circuit simulation as well [6]. Exploiting the *a priori* knowledge about the neglected fill-ins leads to a modified version of the Markowitz algorithm. Because the MA27 implementation does not explicitly update the sparsity pattern at each stage it is not suitable for this reordering variant. More sophisticated data structures that are tuned to the modified Markowitz reordering are described in [11].

Matrix	Dimension orignal/reduced	First Approach [s] orig./red.	MA27 [s] orig./red.
CCP	89556/89378	519/549	750/12
CIRC2A	482969/482963	26788/11396	64901/24
DRA82	138679/138678	1120/936	1809/34
HCIRCUIT	105676/105676	372/372	3/3
SCIRCUIT	170998/170998	1016/1016	7/7

Tab. 1 Reordering times of minumum degree implementations [PC, 850MHz]

V. Results and Conclusion

We have shown that the minimum degree implementation of Duff and Reid can determine the order of the equations very effectively as long as the matrices comply with certain requirements. It is important that prior to reordering a removal of dense and nearly-dense rows and columns has to be carried out. To sum up, we recommend the usage of the MA27 as a reordering implementation for direct solvers in VLSI circuit simulation. When using the MA27, it is important to keep in mind its peculiarities.

[2]For further details on this topic see the literature, e.g., the chapter *Sparsity and the Optimal Ordering of Circuit Equations* of [9]. All matrix structures arising from our circuit examples are nearly symmetrically structured, see [11].

In conclusion, even for very large circuits the symbolic phase requires in general much less computation time than the solution phase if its implementation details are handled very carefully.

Acknowledgments

The NEC Corp., Tokyo/Japan and St. Augustin/Germany provided examples of matrix structures for this study. Furthermore, the matrix examples HCIRCUIT and SCIRCUIT can be found on the web site of Tim Davis, University of Florida [17]. The authors gratefully acknowledge this support.

References

[1] P. R. Amestoy, T. A. Davis, I. S. Duff, Algorithm 8xx: AMD, an approximate minimum degree ordering algorithm, Technical Report

[2] A. Basermann, U. Jaekel, K. Hachiya, Preconditioning parallel sparse iterative solvers for circuit simulation, Proceedings of the Eighth SIAM Conference on Applied Linear Algebra, The College of William & Mary, Williamsburg, VA, July 15-19, 2003

[3] J. J. Dongarra, F. G. Gustavson, A. Karp, Implementing linear algebra algorithms for dense matrices on a vector pipeline machine, SIAM Review, Vol. 26, No. 1, p. 91-112, 1984

[4] I. S. Duff, J. K. Reid MA27: A set of Fortran subroutines for solving sparse symmetric sets of linear equations, Her Majesty's Stationery Office, AERE R10533, 1982

[5] I. S. Duff, A. M. Erisman, J. K. Reid, Direct methods for sparse matrices, Oxford University Press, 1989, ISBN 0-19-853421-3

[6] K.-M. Eickhoff, W. L. Engl, Levelized incomplete LU factorization and its application to large-scale circuit simulation, IEEE Transactions On Computer-Aided Design Of Integrated Circuits And Systems, Vol. CAD-14, No. 6, p. 720-727, 1995

[7] Chung-Wen Ho, A. E. Ruehli, P. A. Brennan, The Modified Nodal Approach to Network Analysis, IEEE Transactions On Circuits and Systems, Vol. CAS-22, No. 6, p. 504-509, 1975

[8] C. Keller Understanding MA27 - A numerical example, Oxford University Computing Laboratory, Numerical Analysis Group, Report no. 00/21, 2000

[9] V. Litovski, M. Zwolinski, VLSI - Circuit simulation and optimization, Chapman & Hall, 1997, ISBN 0-412-63860-6

[10] H. M. Markowitz, The elimination form of the inverse and its application in linear programming, Management Science, Vol. 3, p. 255-269, 1957

[11] I. Naumann, Sortierverfahren und Datenstrukturen in der VLSI-Netzwerksimulation, Doctoral thesis, University at Kiel, 2003, ISBN 3-8322-2390-8

[12] M. Nordhausen, Effizienzuntersuchungen am Minimum-Degree-Algorithmus MA27 für die VLSI-Schaltungssimulation, Student research project, University at Kiel, 2003

[13] D. E. Rose, R. E. Tarjan, Algorithmic aspects of vertex elimination on directed graphs, SIAM J. Appl. Math., Vol. 34, No. 1, p. 176-197, 1978

[14] P. Sadayappan, V. Visvanathan, Efficient sparse matrix factorization for circuit simulation on vector supercomputers, IEEE Transactions On Computer-Aided Design Of Integrated Circuits And Systems, Vol. CAD-8, No. 12, p. 1276-1285, 1989

[15] M. Yannakakis, Computing the minimum fill-in is NP-complete, SIAM J. Alg. Disc. Meth., Vol. 2, No. 1, p. 77-79, 1981

[16] `http://www.cse.clrc.ac.uk/nag/hsl/hsl.shtml`, Website of the Computational Science and Engineering Department, Council for the Central Laboratory of the Research Councils (CCLRC) of the United Kingdom

[17] `http://www.cise.ufl.edu/~davis/sparse/`, Website of the Sparse Matrix Collection, Tim Davis, University of Florida