MULTI-PROCESSOR ARCHITECTURES FOR SOLVING SPARSE LINEAR SYSTEMS. APPLICATION TO THE LOAD FLOW PROBLEM

A. Gómez & L. G. Franquelo

Dpto. Ingen. Eléctrica, Electrónica y Automática
E.T.S. Ing. Industriales.
Avda. Reina Mercedes, 41012 Sevilla

Abstract. Three heuristic algorithms for solving a cluster problem associated with the tearing of a symmetric matrix are presented. Based on these partitioning procedures, a method for the parallel solution of the fast decoupled load flow has been developed, although it is also suitable for the parallel solution of any linear equations system.

Experimental results of applying such algorithms on several test systems have been obtained using a multiprocessor architecture.

Keywords: Multiprocessing Systems, Large Scale Systems, Parallel Processing, Microprocessors, Load Flow, Power Systems Analysis, Iterative Methods.

INTRODUCTION

In recent years, considerable effort has been directed towards the parallel solution of linear equations, due mainly to advances in multiprocessor and array processor computer architectures (Pottle, 1980).

Single instruction multiple data stream (SIMD) processors, which attain parallelism at the instruction level (e.g., vector and/or pipelined operations), have been used to solve the linear equations describing power system networks. References (Pritchard, 1982; Hulskamp, 1982) describe respectively the solution of load flow problem and contingency analysis through this approach.

An alternative type of parallel processor, the multiple instruction multiple data stream (MIMD) processor, can make use of parallelism at other than the instruction level. Arnold (1983) and Fong (1978) give examples of transient stability problem solution.

Following Arnold (1983) two kinds of parallel algorithms can be distinguished. One, which will be referred to as block schemes, is to divide the matrix of interest into blocks, for example, Fong (1978) and Wallach (1980). The number of blocks is simply related to the number of processors. An alternative group of methods exploit the parallelism which is shown to exist when considering each individual element substitution during the solution. These methods shall be referred to as elemental schemes. Wing (1980) gives a theoretical model from which the operations that can be done in parallel are identified, and the absolute minimum completion time and lower bounds on the minimum number of processors required to solve the equations in minimal time can be found. A more practical approach is given in Arnold (1983) where the management overhead is considered and hardware developed is described.

The former group usually requires that the network or system be partitioned into subsystems (i.e., clusters). The optimal network decomposition for a parallel processing system is the bordered block diagonal form (BBDF) ideally composed of equally sized sub-blocks and a minimum sized border.

Elemental schemes need a dynamic method of task allocation among processors, taking account of the actual execution rates of tasks (i.e., task schedulling). The number of synchronizations required is higher than with block schemes.

This paper describes the development of an algorithm based on a block scheme, applied to the solution of the fast decoupled load flow (FDLF) (Stott, 1974.a). Although the detailed description is restricted to two blocks, the ideas for extending it to any number of blocks are given.

Three cluster algorithms for tearing networks are presented below. These algorithms are intended to minimize the number of interconnection nodes within certain limits, although they can also minimize the number of interconnection branches. Ogbuobiri (1970) and Sangiovanni (1977) describe other heuristic algorithms based on the concept of a contour tableau. This problem may not be faced theoretically. In fact, there are very few papers dealing with it.

Afterwards, the application of these algorithms to the parallel solution of the FDLF is dealt with, as well as numerical results of applying them to several test systems.

I. PARTITIONING ALGORITHMS

As has been mentioned, a block-type parallel solution of a linear system of equations requires that the matrix be partitioned accordingly.

In this section three clustering algorithms will be described which will be later used for the parallel solution of the load flow problem.

Consider two disjoint components A and B of a graph (Fig. 1). What is required is to find those components which minimize either the set of nodes of A which are adjacent to B (X) or the set of nodes of B which are adjacent to A (Y). When either of these sets is minimum, A and B are clusters of the graph.

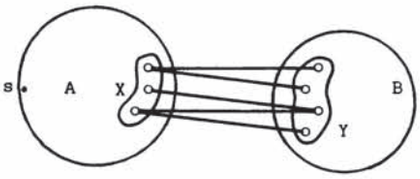


Fig. 1 Two disjoint components of a graph. The bordering nodes are shown.

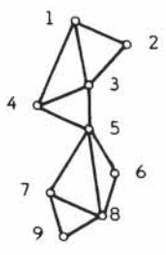
Almost all practical clustering algorithms use the concept of the "contour tableau" which consists of an array of three columns. In the leftmost column, the nodes of the graph are consecutively stored following a predetermined strategy. Referring to Fig. 1, the set A comprises the nodes which have been already included in this column, beginning with the node s. Each time a node is included in this column, the set of nodes X or Y is updated (not reevaluated from scratch) and included in the middle column. The cardinality of this set constitutes the rightmost column. Depending on which set is adopted (X or Y), two tableaus are possible, though it is easy to see that reversing the order of the nodes in the leftmost column, and interchanging X and Y, both tableaus are equivalent. So, the choice of either of them is a matter of programming convenience.

A simple example of a contour tableau based on the set X is shown in Fig. 2.

In the construction of this tableau, there are only two places where choices are made : When an initial iterating node is adopted and when choosing the next iterating node. These degrees of freedom lead to different algorithms.

Sangiovanni (1977) proposed a minimum degree node as the starting node. Assume A is the set of nodes already considered, then the next node chosen is that of Y which makes fewer nodes of B-Y become nodes of Y. Hence, this algorithm looks for a local minimization of the set Y at each step. The adopted strategy is the same as the one proposed by King (1970) for profile minimization purposes. This fact suggests that any algorithm intended for profile and/or band reduction may be useful to solve the cluster problem. The exploitation of this idea leads to the algorithms described below, in ascending complexity.

A minimum degree node is not always the best initial iterating node. To understand this assertion, see the comments made by Sangiovanni (1977) concerning this point. In this paper, a solution to this problem is proposed consisting in taking one of the extremes of a graph's diameter as the starting node. In order to find such extremes a simple algorithm proposed by Gibbs (1976) is adopted. The following algorithms use this procedure.



Node	X	|X|
5	5	1
6	5,6	2
8	5,8	2
7	5,8,7	3
9	5	1
4	5,4	2
3	4,3	2
1	3,1	2
2	-	0

Fig. 2 An example showing the contour tableau.

Algorithm 1. The contour tableau is built with the set X (Fig. 1). The leftmost column of the tableau is filled as follows: Once the starting node has been identified its adjacent nodes are numbered. The algorithm proceeds by numbering the pending nodes adjacent to the lowest numbered node, and so on until the required number of nodes has been included in the tableau.

This process yields a diagonal band ordering since it constitutes a simplified version of the Cuthill-Mckee (1969) algorithm in which the degree of each node is ignored. Whith an adequate storing of the sparse structure of the matrix this process is very fast.

Algorithm 2. Except for the identification of the starting node the algorithm is the same as the one proposed by Sangiovanni (1977) which was previously described. The results obtained are in general better with such choice of the starting node. For instance, examples 1 and 5 in that paper are now optimally partitioned.

Algorithm 3. It is a refinement of Algorithm 2. The next node to be iterated (i.e., included in the set A) is not only looked for among nodes of Y but all nodes of the set B are tried. This usually leads to a further reduction of |Y|, although the computational cost is greatly increased. This strategy was proposed by Levy (1971) in the context of profile minimization.

By using either of the three described methods of building the contour tableau the overall partitioning algorithm is as follows:

Let
- N: number of nodes of the graph.
- b: number of desired diagonal blocks. In this case b equals the number of available processors (b>1).
- s: searching range of the minimum along the contour tableau, in percent of N.
- n: number of nodes already included in the tableau (initially set to zero).
- c: number of interconnection nodes already identified (initially set to zero).
- Z: the set upon which the contour tableau is built (either X or Y depending on the adopted method).

For k=1 to b-1 perform the following steps:

1) Compute the bounds L and U between which a minimum |Z| is going to be looked for, by the expressions:

$$L = n + \frac{N-n}{b-k+1} - \frac{sN}{200}$$

$$U = L + \frac{sN}{100}$$

2) From i= n+1 to i=U build the contour tableau.

3) Find i_k^o with the minimum |Z|, such that $L \leq i_k^o \leq U$.

4) The nodes belonging to Z are interconnection nodes. Set c= c+ |Z|.

5) The set of nodes included between n+1 and i_k^o in the leftmost column of the tableau, excluding those belonging to Z, forms the k-th block. The value n must be updated to $n=i_k^o$.

Finally, b-th block is given by the remaining

Table I. Ordering times.

No-	Ordering Times (sec.)					
des	A1-2	A2-2	A3-2	A1-3	A2-3	A3-3
118	0.25	0.35	1.63	0.30	0.39	1.89
175	0.42	0.57	3.42	0.50	0.66	4.21
265	0.73	1.36	8.05	0.86	1.53	9.59
293	0.83	1.13	9.49	0.97	1.33	11.26
383	1.51	2.26	16.22	1.67	2.56	19.00
448	1.99	3.12	21.64	2.12	3.14	26.16
596	2.94	6.86	38.38	4.44	7.59	50.42
661	3.26	7.16	46.47	4.54	7.99	53.76

Ai-j: Algorithm i for j clusters

nodes, i.e., the nodes not yet included in the tableau.

Each time step 2) is reached a new endpoint of a diameter must be found. As the graph may be disconnected, the biggest connected component is investigated.

Once the clusters and the border have been identified, the minimum degree algorithm (Tinney, 1973) is applied to each cluster in order to avoid an excessive fill-in during the factorization process.

Table I shows the computation times when the algorithms are applied to some electrical networks. Part of these data (two clusters) are plotted in Fig. 3 versus the product Nodes x Branches. From this figure is apparent that A-1 is the fastest while A-3 is extremely slow. The times spent by A-3 are comparable or even greater than those required by the Load Flow itself (see below). This is the reason why A-3 will not be taken into account from now on, though it is potentially useful in other applications.

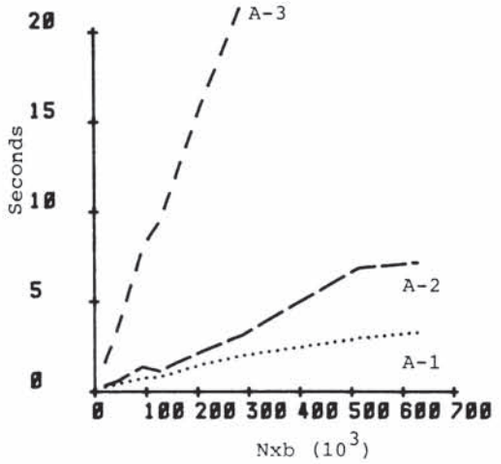


Fig. 3 Computer time spent versus N.b when two clusters are wanted.

II. PARALLEL SOLUTION OF THE FDLF

In this section the parallel solution for each stage of the FDLF (Stott, 1974.a) will be described, assuming for simplicity that we are only interested in two diagonal blocks and, hence, the matrices are partitioned as shown in Fig. 4. Since the matrices Y_{bus}, B' and B" are symmetric, only the upper half is stored and dealt with. A sparse matrix package, based on an efficient random storage with row pointers, has been developed in order to implement the next processes.

The FDLF process requires the repeated solution of

$$\begin{bmatrix} A_1 & 0 & A_2 \\ 0 & A_3 & A_4 \\ A_2^t & A_4^t & A_5 \end{bmatrix} \quad \begin{bmatrix} U_1 & 0 & U_2 \\ 0 & U_3 & U_4 \\ 0 & 0 & U_5 \end{bmatrix} \quad \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

Fig. 4 Partitioned matrices and vectors.

a linear system Ax=b, where A may be either B' or B", and b may be either the mismatch vector $\Delta P/V$ or $\Delta Q/V$. The unknown vector x gives the voltage corrections (phase or module). The corresponding matrix A is factored into $A=U^t.U$ (Fig. 4) before the iterative process. A block scheme must perform the following operations:

$$\text{P1}\begin{cases} U_1^t U_1 = A_1 & (1) \\ U_1^t U_2 = A_2 & (2) \end{cases}$$

$$\text{P2}\begin{cases} U_3^t U_3 = A_3 & (3) \\ U_3^t U_4 = A_4 & (4) \\ U_5^t U_5 = A_5 - U_2^t U_2 - U_4^t U_4 & (5) \end{cases}$$

Operation (1) is the factorization of the diagonal block A_1. Using U_1, U_2 may be computed from (2) by performing the forward elimination on the columns of A_2. Of course, both processes are simultaneously done. While one processor (P1) is carrying out the operations indicated by (1) and (2), the other (P2) is doing the same with (3) and (4). Finally, one of them (P2) must compute U_5 from (5).

The forward elimination process, $U^t y=b$, is somewhat similar:

$$\text{P1}\begin{cases} U_1^t y_1 = b_1 & (6) \end{cases}$$

$$\text{P2}\begin{cases} U_3^t y_2 = b_2 & (7) \\ U_5^t y_3 = b_3 - U_2^t y_1 - U_4^t y_2 & (8) \end{cases}$$

The operations indicated by (6) and (7) are done in parallel by the two processors. Elimination process is finished when one of the processors (P2) performs (8).

The unknown vector x is obtained from y during the back substitution process, Ux=y:

$$\text{P2}\begin{cases} U_5 x_3 = y_3 & (9) \\ U_3 x_2 = y_2 - U_4 x_3 & (10) \end{cases}$$

$$\text{P1}\begin{cases} U_1 x_1 = y_1 - U_2 x_3 & (11) \end{cases}$$

Previous solution of equation (9) gives x_3, which is then used by both processors to solve respectively (10) and (11).

Each half-iteration of the FDLF must solve either the active or the reactive subproblem. Computation of mismatch vectors takes about 70% of the time required at each iteration. Hence, careful attention must be devoted to the parallel solution of this stage.

Figure 5 shows the upper half of Y_{bus}, and the accordingly partitioned vector $\Delta P/V$. Elements of submatrix Y_1 contribute only to $(\Delta P/V)_1$ while elements of Y_2 contribute both to $(\Delta P/V)_1$ and

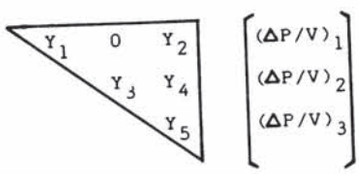


Fig. 5 Partitioned Y-bus and $\Delta P/V$.

$(\Delta P/V)_3$. In the same way, elements of Y_3 only affect $(\Delta P/V)_2$ whereas elements of Y_4 contribute to $(\Delta P/V)_2$ and $(\Delta P/V)_3$. Using a conventional notation:

$$(\Delta P/V)_1 = (P^{esp}/V)_1 - (P^{cal}/V)_{C_1} - (P^{cal}/V)_{C_2} \quad (12)$$

$$(\Delta P/V)_2 = (P^{esp}/V)_2 - (P^{cal}/V)_{C_3} - (P^{cal}/V)_{C_4} \quad (13)$$

$$(\Delta P/V)_3 = (P^{esp}/V)_3 - (P^{cal}/V)_{C_2,C_4,C_5} \quad (14)$$

where C_i means "contribution of Y_i".

After equations (12) and (13) are solved in parallel, one processor (P2) solves (14). Alternative and probably improved schemes are still possible.

Sequence coordination must be performed by means of properly located flags. This coordination is easier to achieve than with other parallel schemes (e.g. elemental schemes) and it is not so complex as may be thought. For example, the iterative process only requires synchronization in five places. It is clear from the explanation that certain matrices and vectors must be accesible to both processors.

The extension of this parallel solution procedure to more than two processors is quite an easy task: One additional processor for each added block is required. The code in these processors is practically the same as in processor P1. The code of P2 must be expanded in order to take account of the newly created off-diagonal blocks. Finally, additional flags must be introduced to synchronize the whole process correctly.

HARDWARE CONFIGURATION

To test the proposed algorithms, a low cost architecture has been adopted consisting of 16 bits microprocessors connected to a standard VME bus (Fig. 6). A global memory acts as the communicating element between processors, and stores the common items.

Each processor is a single card based on the MC68000 chip running at 8Mhz including up to 128 Kbytes of local memory. The arithmetic processor was added because of this particular application.

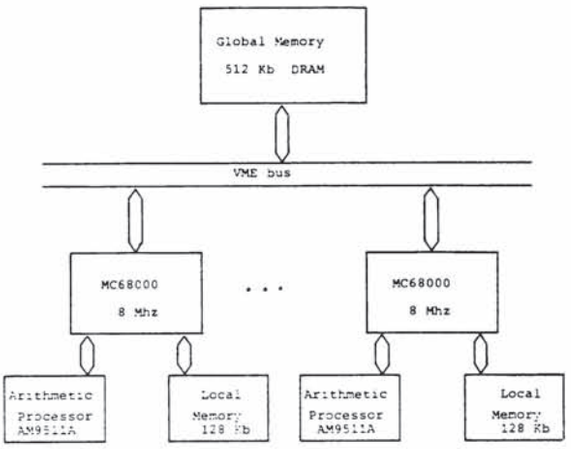


Fig. 6 Hardware configuration.

COMPARATIVE RESULTS

In order to allow comparisons, both the sequential and parallel FDLF have been programmed, with the same hardware and assembler language.

Table II summarizes the results obtained applying the formerly mentioned algorithms to eight networks ranging from 118 to 661 nodes. Each parallel solution has been implemented both with two and three processors. The upper-leftmost columns present the time required by the five programmed methods to carry out one iteration. The absolute time does not have a real interest due to the arithmetic processor incorporated (AM9511). It runs at 2 MHz and its 8-bit external bus introduces a large overhead. Currently, there is more powerful commercial VME hardware available, based on the MC68020 plus the arithmetic coprocessor, which runs at a higher speed. If it were used, the absolute times would be drastically reduced, since the performance of one of these processors is comparable to 1.5 times that of the VAX 11/780 (Zorpette, 1985).

The bottom-right corner points out the number of iterations required to meet the convergence criterion (10^{-3} or 10^{-4} pu.). The results shown refer to unadjusted solutions, because we are mainly interested in the basic algorithm. Nevertheless these adjustments are essential in any practical application. Clearly, the power flow adjustments would not have an appreciable effect on the proposed parallel logic, if they are done trough a method especially developed for the FDLF (e.g. the error feedback adjustment proposed by Stott (1974a,b)). The main effect would be a greater number of iterations to the same extent as in sequential FDLF.

It becomes apparent that the number of required iterations is higher than the number of those

Table II. Results obtained with some test systems.

No-	Time per Iteration (sec.)					Efficiency (%)				Size of Border				Semi-
des	Seq.	A1-2	A2-2	A1-3	A2-3	A1-2	A2-2	A1-3	A2-3	A1-2	A2-2	A1-3	A2-3	Iters
118	0.46	0.31	0.27	0.24	0.21	75.3	86.4	63.9	72.6	6	3	6	4	9
175	0.72	0.41	0.42	0.38	0.33	87.2	85.6	63.2	72.2	3	3	5	4	13
265	1.16	0.75	0.83	0.84	0.68	77.1	69.5	45.8	56.7	11	13	15	14	19
293	1.19	0.68	0.68	0.56	0.56	87.7	87.7	70.5	70.8	3	3	4	3	18
383	1.72	1.20	1.05	0.95	0.91	71.7	82.0	60.5	63.2	12	6	18	10	23
448	1.97	1.56	1.22	1.40	0.96	63.2	80.5	47.0	68.5	21	11	20	11	28
596	2.82	1.77	1.88	2.87	1.83	79.5	75.0	32.7	51.3	18	11	34	23	19
661	3.06	1.90	1.93	1.90	1.85	80.5	79.3	53.6	55.1	13	9	25	17	17

Seq: Sequential FDLF, Ai-j: Algorithm i for j Processors.

required with the IEEE test systems, due to the small X/R ratio and to the strong loading condition of the tested networks.

The border size which appears in Table II does not include the slack node. This size is not the minimum attainable with the algorithms, but the one which provides the minimum time for the whole process. For instance, A-1 yields a border of 2 nodes, with the 118 node system, but the unequal size of resultant blocks (47,68) produces longer times than the data shown in Table II, which correspond to cluster sizes of 57, 54 and 6 nodes obtained with a narrower searching range.

The proposed factorization process behaves irregularly, though its results are not shown in Table II. In any case, the time required for the factorization only represents a small fraction of the total (about 3%).

Undoubtedly the most important time is that which is needed to complete the iterative process (90% of the total), on which the attention will be focused.

Figures 7 show the comparative results of the iterative process in the five cases mentioned before. Figure 7.a shows the absolute time per iteration; Fig. 7.b points out the efficiency of this process, defined as 1/(r.n), where r is the ratio parallel time/ sequential time of the iterative process and n is the number of involved processors. The meaning of this parameter is the average time that each processor has been useful.

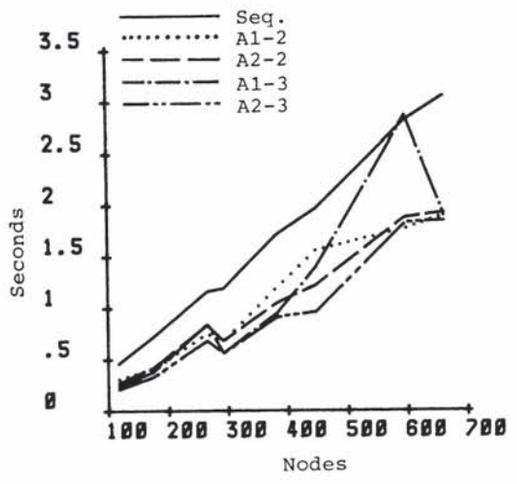


Fig. 7.a Time per iteration.

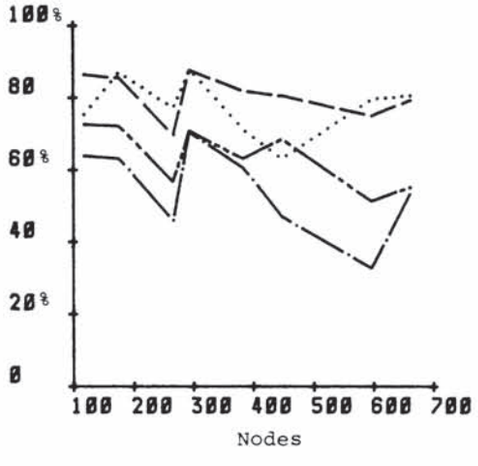


Fig. 7.b Efficiency of processors.

Both A-1 and A-2 provide good results for 2 processors. In this case the efficiency is around 80%, with a minimun of 63% and a maximum of 88%. It seems that A-2 behaves better than A-1, though not in all the tested systems. It must be remembered that the ordering time of A-2 is always greater than that of A-1, which means that sometimes the situation is different with respect to the total time (which may be computed as the product of the number of iterations by the time per iter. plus the ordering time).

The first clear thing for 3 processors is the disparity of results and their global tendency to be worse than with 2 processors. The disparity is justified by the two kinds of tested networks. On the one hand, there are systems (175 and 293 nodes) created by weakly joining two or more real systems. In these systems, the clusters are correctly identified and the efficiency is quite acceptable (72%, 71%). On the other hand, the real Spanish systems are composed of very meshed and strongly connected subnetworks, which means that there are no natural clusters to be found. Hence, the efficiency drops substantially despite the relative small size of the border (see Table II). Three main reasons justify this loss of efficiency:

1) Some vectors and matrices must be stored in the global memory whose access time is longer.
2) The presence of the border, where the parallelism does not apply.
3) And the most important one, the gradual loss of sparsity that results when the node ordering is not that of minimum degree algorithm (Tinney, 1973).

CONCLUSIONS

In this paper three automatic ways of partitioning a sparse matrix into blocks of similar size with a small interconnection are presented. Based on these partitions, a method for the parallel solution of the FDLF has been developed and tested on several examples via a low cost standard microprocessor architecture.

These heuristic partitions have proved to be efficient, even for systems without natural clusters yielding a border size of 10% of the nodes in the worst case, either with 2 or 3 blocks.

The performance of the parallel solution has been quite good when using two processors. The efficiency in this case is around 80%. For three processors, two types of behavior have appeared: In those systems with natural clusters the efficiency remains high, while in those strongly connected it drops near 50%.

One of the contributions of the paper is the implementation of the proposed algorithms on a real architecture, applying them to different real networks. Until now little effort has been done in this direction, as the results shown in the literature refer to trivial or very special test systems.

The parallel algorithm may be suitable for those dispatch centers with a dual computer configuration in which the opportunity appears to increase the use of the bystanding machine.

REFERENCES

Arnold C.P., Parr M.I., Dewe M.B. (1983). An Efficient Parallel Algorithm for the Solution of Large Sparse Linear Matrix Equations. IEEE Trans. on Computers C-32, pp 265-272.

Cuthill E., McKee J. (1969). Reducing the Bandwidth of Sparse Symmetric Matrices. Proc. 24th National Conf. ACM. pp 157-172.

Fong J., Pottle C. (1978). Parallel Processing of Power System Analysis Problems Via Simple Parallel Microcomputer Structures. IEEE Trans. on PAS-97, pp 1834-1841.

Gibbs N.E., Poole W.G., Stockmeyer P.K. (1976). An Algorithm for Reducing the Bandwidth and Profile of a Sparse Matrix. SIAM J. N. Anal. vol. 13, pp 236-250.

Hulskamp J.P., Chan S.M., Fazio J.F. (1982). Power Flow Outage Studies Using an Array Processor. IEEE Trans. on PAS-101, pp 254-261.

King I.P. (1970). An Automatic Reordering Scheme for Simultaneous Equations Derived from Network Systems. Int. J. Num. Meth. Eng. Vol. 2, pp 523-533.

Levy R. (1971). Restructuring of the Structural Stifness Matrix to Improve Computational Efficiency. Jet Propulsion Lab. Tech. Rev. vol. 1, pp 61-70.

Ogbuobiri E.C., Tinney W.F., Walker J.W. (1970). Sparsity Directed Decomposition for Gaussian Elimination on Matrices. IEEE Trans. on PAS-89, pp 141-150.

Pottle C. and Others. (1980). Proceedings of a Computer Hardware Workshop. Proc. of SIAM. Electric Power Problems: The Mathematical Challenge, pp 499-514, Philadelphia.

Pritchard R., Pottle C. (1982). High-Speed Power Flows Using Attached Scientific ("Array") Processors. IEEE Trans. on PAS-101, pp 249-253.

Sangiovanni-Vincentelli A. and Others. (1977). An Efficient Heuristic Cluster Algorithm for Tearing Large Scale Networks. IEEE Trans. on CAS-24, pp 709-717.

Stott B., Alsac O. (1974). Fast Decoupled Load Flow. IEEE Trans. on PAS-93, pp 859-867.

Stott B. (1974). Review of Load-Flow Calculation Methods. Procee. IEEE vol. 62, pp 916-924.

Tinney W.F., Meyer W.S. (1973). Solution of Large Sparse Systems by Ordered Triangular Factorization. IEEE Trans. on Automatic Control AC-18, pp 333-345.

Wallach Y., Konrad V. (1980). On Block-Parallel Methods for Solving Linear Equations. IEEE Trans. on Computers C-29, pp 354-359.

Wing O., Huang J.W. (1980). A Computational Model for Parallel Solution of Linear Equations. IEEE Trans. on Computers C-29, pp 632-638.

Zorpette G. (1980). The Beauty of 32 bits. IEEE Spectrum, pp 65-71 Sep.