PREPRINTS

Fourth IASTED International Conference

HIGH TECHNOLOGY IN THE POWER INDUSTRY
POWER HIGH TECH'89

Valencia, Spain

Edited by

C. Alvarez

and

P. Albertos

UNIVERSIDAD POLITECNICA DE VALENCIA

ADVANCED COMPUTER TECHNIQUES IN POWER SYSTEMS

STEADY STATE ANALYSIS

IASTED Power High Tech'89
Valencia, Spain, 1989

Three Methods for the Parallel Solution of a Large, Sparse System of Linear Equations by Multiprocessors

A. Torralba*, A. Gómez† and L.G. Franquelo*

E. T. S. Ingenieros Industriales
Av. Reina Mercedes, s/n
41012 Sevilla, Spain

* Departamento de Ingeniería Electrónica, de Sistemas y Automática
† Departamento de Ingeniería Eléctrica

Abstract

In this paper three methods for the parallel solution of large–scale, sparse, linear equation systems are presented and compared. Both, factorization of the matrix and forward–backward processes, have been considered. A multiprocessor architecture has been simulated and results are presented corresponding to eight examples taken from power systems field, ranging from 117 to 660 equations.

Keywords: Sparse Systems, Multiprocessors, Parallel Processing, Simulation.

1 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.

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 linear equations in different engineering fields, e.g., in power system analysis [1]–[3].

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. References [4] and [5] give examples in the same engineering field through this approach.

Following [4] two kinds of parallel algorithms can be distinguished. One, which will be referred to as block schemes, divide the matrix of interest into blocks, for example, [5]–[6]. 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. Reference [7] 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 [4] where the management overhead is considered and hardware developed is described.

The former group (block schemes) requires the network or system to be partitioned into subsystems (i.e., clusters). A possible 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, accounting for the actual execution rates of tasks (i.e., task scheduling). The number of synchronisations required is usually higher than with block schemes.

In this paper a third scheme is presented, termed the hybrid scheme, which tries to gather the best properties of both kinds of parallelism. This scheme also requires a partitioning of the system matrix. Unlike block methods, the border is solved in parallel by means of an elemental scheme.

Three algorithms, each representing one of the three schemes above mentioned, are described in this paper and applied to the solution of eight test systems, whose sizes range from 117 to 660 equations. A simple multi–microprocessor architecture with a single bus has been simulated to implement the algorithms.

2 Description of the Algorithms

Consider a large, sparse system of linear equations given by:

$$Ax = b \tag{1}$$

For the sake of simplicity, assume that the system matrix is symmetric and, hence, only the upper half is stored and dealt with. Anyway, the algorithms described below, may be easily extended to cover the non–symmetric case.

2.1 Block scheme

A method for partitioning the involved matrices into blocks is needed if a block scheme is going to be used for the parallel solution of (1). A few papers deal with this problem (e.g., [8]–[10]) and the algorithm in [10] has been used here because of its good performance.

Suppose, without loss of generality, that only two processors are available. The matrices are, then, partitioned into two equally–sized diagonal blocks and a minimum border (figure 1).

Before the forward/backward process, the matrix must be factored into $A = U^tU$ (figure 1) as follows:

$$U_1^tU_1 = A_1 \tag{2}$$

$$U_1^tU_2 = A_2 \tag{3}$$

$$U_3^tU_3 = A_3 \tag{4}$$

$$U_3^tU_4 = A_4 \tag{5}$$

$$U_5^tU_5 = A_5 - U_2^tU_2 - U_4^tU_4 \tag{6}$$

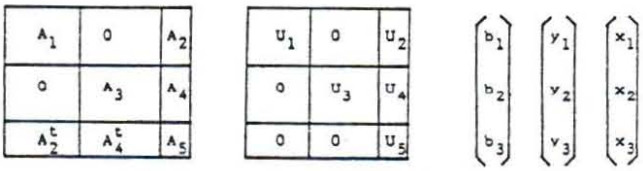

Fig. 1. Partitioned matrices and vectors.

Operation (2) is the factorisation of the diagonal block A_1. Using U_1, U_2 may be computed from (3) by performing the forward elimination on the columns of A_2. Of course, both processes are simultaneously done. While one processor is carrying out the operations indicated by (2) and (3), the other is doing the same with (4) and (5). Finally, one of them must compute U_3 from (6).

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

$$U_1^t y_1 = b_1 \tag{7}$$

$$U_3^t y_2 = b_2 \tag{8}$$

$$U_3^t y_3 = b_3 - U_2^t y_1 - U_4^t y_2 \tag{9}$$

The operations indicated by (7) and (8) are done in parallel by the two processors. Elimination process is finished when one of the processors performs (9).

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

$$U_5 x_3 = y_3 \tag{10}$$

$$U_3 x_2 = y_2 - U_4 x_3 \tag{11}$$

$$U_1 x_1 = y_1 - U_2 x_3 \tag{12}$$

Previous solution of equation (10) gives x_3, which is then used by both processors to solve respectively (11) and (12).

Figure 2 shows one possible schedule using two processors. In this figure, block 2 is assumed to be greater than block 1 and processor 1 takes account of the non-parallel part of the algorithm (i.e., equations (6), (9) and (12)).

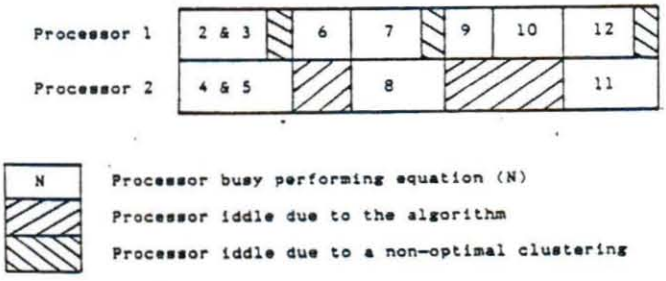

Fig. 2. A possible schedule with two processors.

2.2 Elemental scheme

Some papers deal with elemental parallelism when applied to solve linear equations ([4], [7]). The main drawback of elemental schemes is synchronisation overhead. An elemental scheme with reduced overhead is adopted in this paper [11].

Let us consider a symmetric matrix whose sparsity structure is shown in figure 4.

Operations may be classified into two distinct types:

1. Inversion. $a_{ii} = 1/a_{ii}$
2. Update. $a_{ij} = a_{ij} - a_{ki}a_{kk}a_{kj}$, $j \geq i > k$

One possible sequence of operations to factorize the matrix is listed in Table 1.

It is possible to build a graph, named task graph [7], containing all the precedence relations that exist among elemental operations. However, the process to obtain this graph is complex, processors scheduling is not trivial and introduces overhead.

Consider now that these operations might be grouped into blocks as shown in Table 1. A block consists of the inversion of a diagonal element and the update operations involving this inverted element.

Precedence rules among these blocks are obtained from the simple directed path graph instead of from the more complex task graph, reducing synchronisation overhead. The path graph of the system

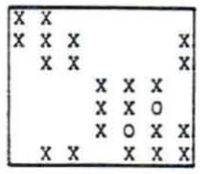

Fig. 3. Sparse structure of the matrix.

matrix may be obtained following [12]; the path for node (block) 1 is depicted in figure 4a. Figure 4b represents the full path graph. A compact description of this graph is achieved, either with the vector NEXT or with the list structure PRED (figure 4c). NEXT(i) contains the successor, and PRED(i) the predeccessors, of node i in the path.

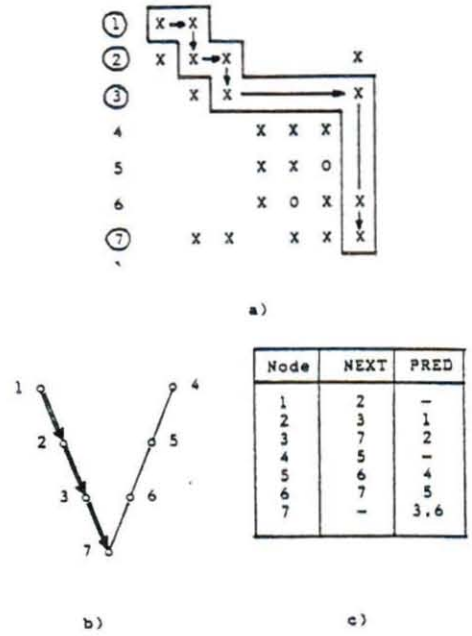

Fig. 4. Path Graph.

Block 1	1	$a_{11} = 1/a_{11}$
	2	$a_{22} = a_{22} - a_{12}a_{11}a_{12}$
Block 2	3	$a_{33} = 1/a_{33}$
	4	$a_{33} = a_{33} - a_{23}a_{33}a_{23}$
	5	$a_{37} = a_{37} - a_{23}a_{33}a_{37}$
	6	$a_{77} = a_{77} - a_{37}a_{33}a_{37}$
Block 3	7	$a_{33} = 1/a_{33}$
	8	$a_{77} = a_{77} - a_{37}a_{33}a_{37}$
Block 4	9	$a_{44} = 1/a_{44}$
	10	$a_{55} = a_{55} - a_{45}a_{44}a_{45}$
	11	$a_{56} = a_{56} - a_{45}a_{44}a_{46}$
	12	$a_{66} = a_{66} - a_{46}a_{44}a_{46}$
Block 5	13	$a_{55} = 1/a_{55}$
	14	$a_{66} = a_{66} - a_{56}a_{55}a_{56}$
	15	$a_{67} = a_{67} - a_{56}a_{55}a_{57}$
	16	$a_{77} = a_{77} - a_{57}a_{55}a_{57}$
Block 6	17	$a_{66} = 1/a_{66}$
	18	$a_{77} = a_{77} - a_{67}a_{66}a_{67}$
Block 7	19	$a_{77} = 1/a_{77}$

Table I. Operations required to factor the matrix of figure 4.

In a practical implementation an auxiliary integer vector, PEND, is required. Whenever PEND(i) = 0, node i is a *ready node* (that is, the operations in block i are ready to be executed).

In the factorisation process, the initial value of PEND is set by PEND(i) = number of elements in PRED(i).

Each processor carries out the following operations:

1. Search for a *ready node* (PEND(i) = 0).
2. Perform the arithmetic operations of the i–th node.
   (a) Inversion: $a_{ii} = 1/a_{ii}$
   (b) Updates: $a_{ij} = a_{ij} - a_{hi}a_{kk}a_{kj}$, $j \geq i > k$
3. PEND(h) = PEND(h) - 1, where h = NEXT(i).

Forward elimination is similar to factorisation process. Step 2 in the above procedure is substituted by:

(a) $b_i = b_i a_{ii}$
(b) $b_j = b_j - a_{ij}b_i$, $j > i$

In backward substitution, precedence relations are determined reversing the direction of the arcs in the path graph. The initial value of PEND(i) is now initialisated to 1, except for the highest numbered node whose initial value is set to zero (this node has not a successor).

Backward substitution process consists of:

1. Search for a ready node (PEND(i) =0).
2. Perform the operations of the i–th node
   $$b_i = b_i - a_{ii}a_{ij}b_j,\ j > i$$
3. PEND(k) = 0, $\forall k \in$ PRED(i).

In order to avoid conflicts each entry in the matrix has one associated semaphore. When an element is going to be updated the semaphore is set, preventing this element from being accessed by other processors.

Actually, the algorithm used is slightly more complex than the procedure described above, i.e., redundant operations (such as multiple products of type $a_{ki}a_{kk}$ in update operations) are avoided in order to reduce operation time.

This procedure is similar to the one described in [4]. Unlike that, the proposed algorithm derives precedences from the simple path-graph and is more suitable for being applied to the factorization process.

2.3 Hybrid scheme

The hybrid algorithm is as follows:

1. Perform the block partitioning of the matrix. It is preferable to obtain equally–sized blocks in this case, rather than minimum–sized border.
2. Solve each diagonal block in parallel. In case of two clusters, this corresponds to solve eqs. (2), (3), (4) and (5) during factorisation step, eqs. (7) and (8) in forward process and eqs. (11) and (12) in backward process.

   Note that, from an elemental–scheme point of view the process is the same as if each processor performed only the elemental operations of a major branch of the path graph (see figure 5 for three clusters). However, this block solution avoids the synchronization required by the elemental method.

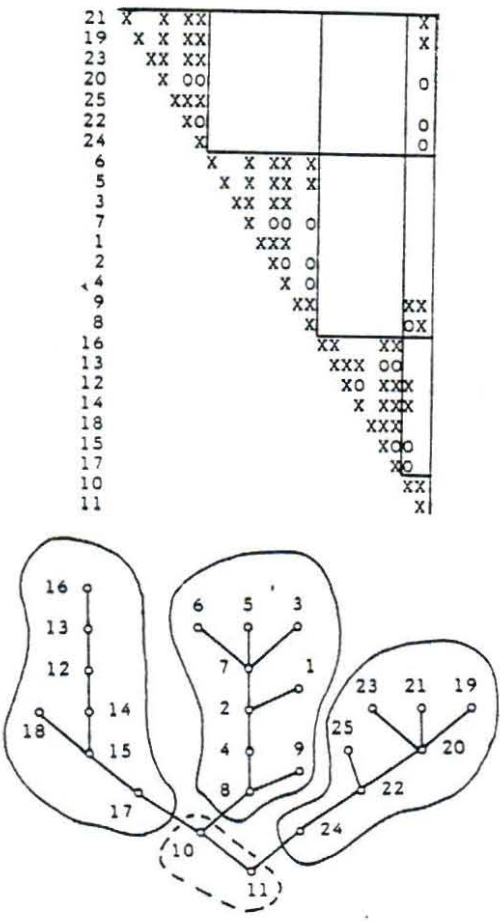

Fig. 5. An example with three clusters.

3. Operations related to the border (fig. 5) are carried out following an elemental scheme. Eqs. (6), (9) and (10) are solved in this way, completing the process.

Performing border operations in such a way relaxes the requirement of a minimum border. This allows to focus attention on finding diagonal blocks of equal size, increasing parallelism in step 2).

3 Practical Implementation

In order to evaluate the algorithms, their performance has been simulated in a VAX computer. In the simulation process, linear systems of equations corresponding to a typical power system analysis problem [13] are actually solved.

Since elemental and hybrid schemes require that all processors may gain partial or whole access to the matrix, a common–memory single–bus architecture has been chosen. In addition, each processor has a local memory and an arithmetic coprocessor (FPU). The CPU instruction set pattern has been taken from a 68000–based VME–bus system.

Different arithmetic operation times have been simulated to find out the effects of CPU relative speed (FPU average operation time to CPU average instruction time ratio) on algorithms' efficiencies. Multiplication and division times are assumed to be equal and 50% greater than that of addition/ substraction.

Like in reference [4], four overhead types appear:

1. Increased number of operations due to a larger fill–in in the factorized matrix as a consequence of special orderings (this applies only to partitioned–based schemes)

2. Searching for *ready nodes* and semaphore overhead (especially notable in elemental schemes)
3. Idle time waiting for a *ready task*.
4. Bus conflicts.

Only the last one has not been considered. Anyway, its effects may be neglected for a reduced number of processors.

The best ordering algorithm proposed in [14] has been adopted in elemental schemes. This yields a shorter mean path length and increases the number of initial *ready nodes*, which is a more suitable graph structure. In addition, the total fill-in is even reduced compared to the minimum degree method.

4 Simulation Results

The algorithms have been tested on eight different power system networks including the IEEE 118-node standard network and other actual spanish systems up to 661 nodes.

The algorithm performance index considered is the efficiency of the parallel process, defined by the ratio $t_1/(nt_n)$, where t_i is the process execution time with i processors.

First of all the forward/backward process will be dealt with. Figure 6 shows the mean of the efficiencies of tested networks versus the number of processors for two different CPU relative speeds. (Floating addition time is equal to 5 CPU instruction cycles in Fig. 6a and equal to 80 CPU instruction cycles in 6b).

It may be noticed that the methods which require the partition

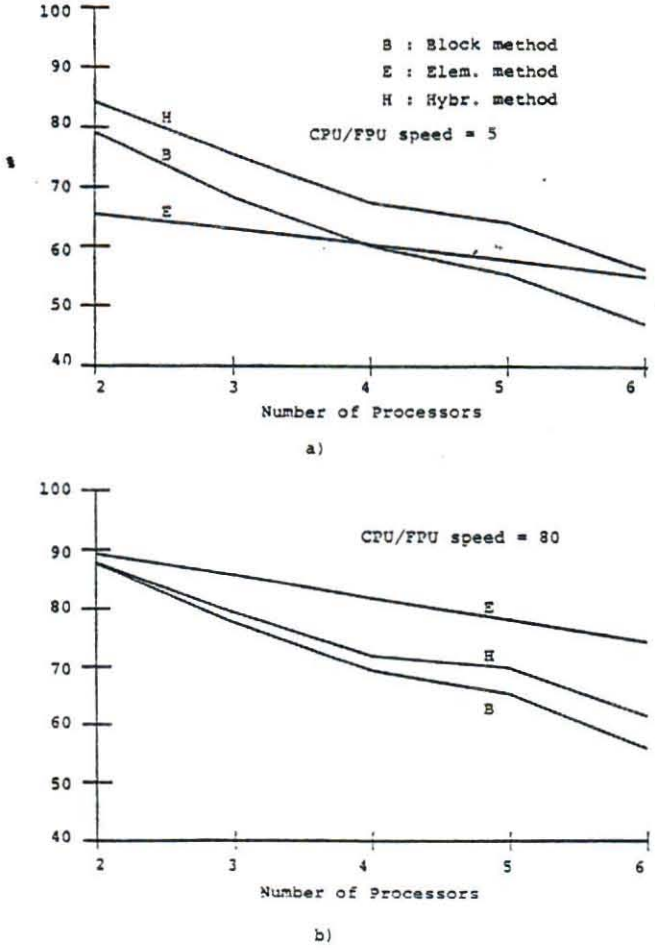

Fig. 6. Forward-Backward process efficiency with different CPU-FPU speed ratios.

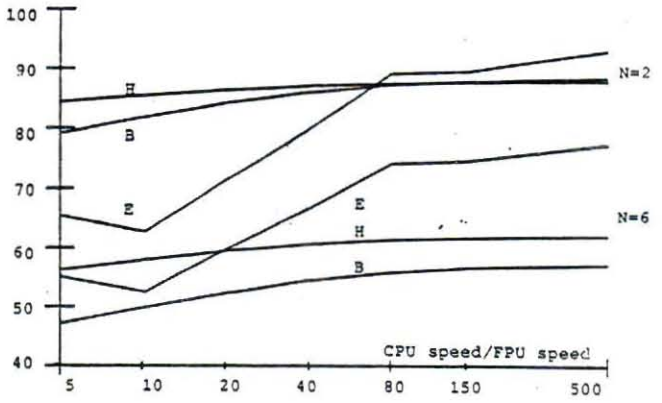

Fig. 7. Forward-Backward process efficiency with two and six processors.

of the matrix are more sensitive to the number of processors because the clustering algorithm performs worse when this number grows.

On the other hand, the elemental scheme's efficiency drops appreciably when a fast FPU is employed. In this case, arithmetic operation times are reduced and the effects of synchronisation overhead are more noticeable. This may be seen clearly in Fig. 7 where the efficiencies versus CPU relative speeds are plotted for 2 and 6 processors. In this sense the hybrid method exhibits an intermediate behavior. It is neither so dependent on the number of processors, nor so influenced by FPU speed.

Fig. 8 is the counterpart of Fig. 7 for the factorization process, illustrating similar results.

The efficiency of the backward/forward process versus system size is shown in Fig. 9 for 2 and 6 processors (FPU add time is 20

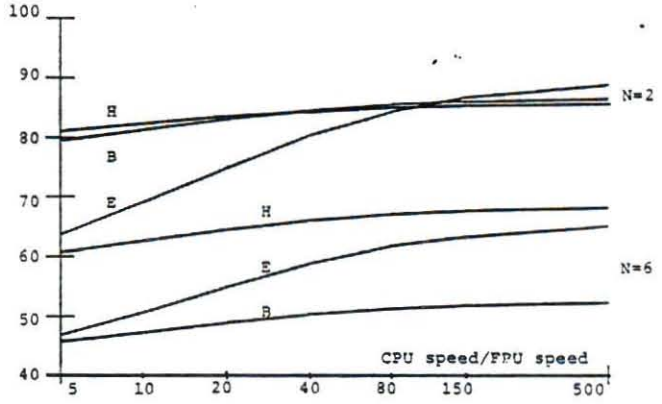

Fig. 8. Factorization process efficiency with two and six processors.

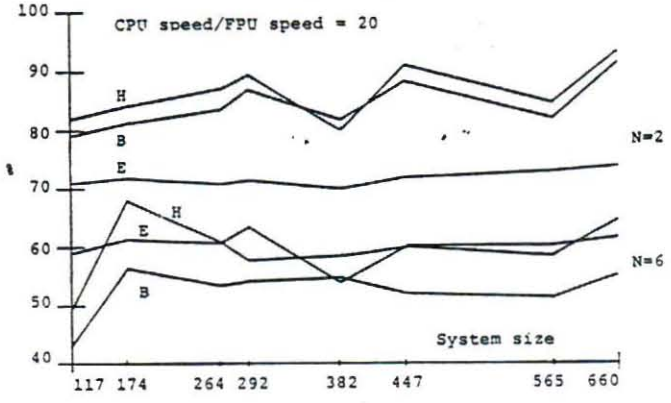

Fig. 9. Forward-Backward process efficiency with two and six processors.

CPU cycles). Experiencies indicate this efficiency is more dependent on the system topology than on the system size. The irregular shape of the curves in this figure obeys to the different nature of the actual power networks used.

5 Conclusions

Three parallel methods for solving large, sparse, linear systems of equations have been presented in this paper. The first one requires the partitioning of the matrix into as many blocks as processors are available. The second algorithm exploits the inherent parallelism existing among elemental operations. Precedence rules are deducted from the simple path graph. Finally a hybrid method is proposed which performs also the partitioning of the matrix but solves the interconnection of the blocks by means of an elemental strategy.

Experimental results reveal:

1. Block scheme is more suitable when the clustering algorithm performs well. This usually happens if a low number of processors is used or natural clusters may be found in the matrix. Furthermore, FPU relative speed does not have an appreciable influence on its behavior.

2. Elemental scheme's performance is quite sensitive to FPU relative speed but allows a larger number of processors to be used.

3. Hybrid method represents a compromise solution which is not so influenced by FPU speed and the number of processors.

Some improvements have been achieved in all cases by using recently-developed ordering algorithms.

6 Acknowledgements

The authors would like to acknowledge the finantial support by the Spanish CICYT, project number PA86-0233-C02-01.

References

[1] Pritchard R., Pottle C., High-speed power flows using attached scientific (*array*) processors. IEEE Trans. on PAS-101, pp. 249-253, 1982.

[2] Hulskamp J.P., Chan S.M., Fasio J.F., Power flow outage studies using an array processor. IEEE Trans. on PAS-101, pp. 254-261, 1982.

[3] Taoka H., Abe S., Takeda S., Fast transient stability solution using an array processor. IEEE Trans. on PAS-102, pp. 3835-3841, 1983.

[4] Arnold C.P., Parr M.I., Dewe M.B., An efficient parallel algorithm for the solution of large sparse linear matrix equations. IEEE Trans. on Computers C-32, pp 265-272, 1983.

[5] Fong J., Pottle C., Parallel processing of power system analysis problems via simple parallel microcomputer structures. IEEE Trans. on PAS-97, pp 1834-1841, 1978.

[6] Wallach Y., Konrad V., On block-parallel methods for solving linear equations. IEEE Trans. on Computers C-29, pp. 354-359, 1980.

[7] Wing O., Huang J.W., A computational model for parallel solution of linear equations. IEEE Trans. on Computers C-29, pp. 632-638, 1980.

[8] Ogbuobiri E.C., Tinney W.F., Walker J.W., Sparsity directed decomposition for Gaussian elimination on matrices. IEEE Trans. on PAS-89, pp. 141-150, 1970.

[9] Sangiovanni-Vincentelli A. et al., An efficient heuristic cluster algorithm for tearing large scale networks. IEEE Trans. on CAS-24, pp. 709-717, 1977.

[10] Gómez A., Franquelo L.G., A new contribution to the cluster problem. IEEE Trans. on CAS-34, pp. 546-552, 1987.

[11] Torralba A., Gómez A., Franquelo L.G., Parallel solution of power system analysis problems. Presented at the IEEE Melecon'87, Rome, Mar. 24-26, 1987.

[12] Tinney W.F., Brandwajn V., Chan S.M., Sparse vector methods. IEEE Trans. on PAS-104, pp. 295-301, 1985.

[13] Stott B., Alsac O., Fast decoupled load flow. IEEE Trans. on PAS-93, pp. 859-867, 1974.

[14] Gómez A., Franquelo L.G., Node ordering algorithms for sparse vector method improvement. IEEE Trans. on Power Systems PWRS-3, pp. 73-79, 1988.