

Application of clustering and factorisation tree techniques for parallel solution of sparse network equations

J. Bialek, PhD, CEng, MIEE
D.J. Grey, BEng, AMIEE

Indexing terms: Diakoptical techniques, Parallel processing, Power system analysis

Abstract: Through the use of diakoptical techniques it is possible to solve the network equations for a power system using a parallel computer architecture. Traditional parallel solution techniques give an unimpressive return on investment as more processors are applied to the problem owing to unacceptably large sequential sections of the algorithm and ill-balanced computational loads. The paper analyses the diakoptical method using the factorisation tree as a tool. New algorithms are proposed for network splitting, node ordering and load balancing between processors. This, in connection with a novel recursively parallel solution algorithm, gives improved speedup over the conventional parallel solution method. Simulation results are presented which confirm the usefulness of the method.

1 Introduction

The solution of large, linear, sparse network equations is a recurrent problem in almost every algorithm in power system analysis, e.g. load flow, state estimation, dynamic simulation, security assessment, etc. The last two applications are time-critical and require a very fast response. Parallel methods can offer a significant speedup but the solution of the linear equations proves to be a major bottleneck seriously hindering the prospect of real-time solution [1]. This paper looks at the way in which the direct, sparsity-oriented method of solution employing one of the factorisation techniques [2-4], can be implemented on cheap, off-the-shelf, message-passing, multiple instruction multiple data (MIMD) parallel processing systems, e.g. the inmos transputer family.

It was realised from the beginning [2] that the simplest approach to a parallel solution is by clustering a large network into several smaller, loosely connected subnetworks. This approach has been extensively analysed [5-7] and various tearing schemes have been proposed [3, 8, 9]. The introduction of the factorisation tree concept first for Cholesky factorisation [10] and then in a power systems context [11] provided, in a compact

form, a very powerful tool for looking at the parallelism in factorisation and forward/backward (F/B) substitution by describing the precedence relationship between the elimination of nodes. From this the new approach of sparse matrix inverse factors [11, 12] has been developed and the traditional diakoptics-based methods have been enhanced [13, 14].

The research reported in this paper has been spurred on by the fact that the highest reported parallel speedup did not exceed a value of about three or four even when many processors were used on large networks.* We first try to find the reason for such an unimpressive return on investment of processing power and attempt to enhance the traditional, diakoptics-based approach with the additional insight given by the factorisation tree. We prove that factorisation tree-based clustering does not deteriorate sparsity of the factor matrices and propose a modified node ordering algorithm. We look at how to tear the network in an optimal way from the point of view of scheduling tasks to individual processors. Finally, we propose a novel recursively parallel solution method which reduces the sequential part of the algorithm.

2 Standard method of parallel solution

A set of linear network equations is usually of the form

$$Ax = b \quad (1)$$

where A is a nonsingular, admittance-type, sparse, diagonally-dominant, symmetric (or incidence symmetric) matrix of the order n ; x is the unknown solution vector and b is the given independent vector, usually full. The standard method of solution is to perform a sparsity-oriented LU decomposition of A followed by forward/backward (F/B) substitution.

Assume that the network, described by A , has been divided into m independent subnetworks connected by tearing (or cut-set) nodes. If the nodes within each subnetwork are numbered consecutively, and the cut-set

* The exception is the sparse inverse factors method [12] suitable for a shared-memory, common-bus type architecture rather than the distributed MIMD. On the other hand, the research reported in Reference 1 shows that the several massive global communication steps required again reduce the speedup gain to about four.

© IEE, 1994

Paper 1289C (P11), first received 3rd November 1993 and in revised form 5th April 1994

The authors are with the School of Engineering, University of Durham, Science Laboratories, South Road, Durham DH1 3LE, United Kingdom

The research reported has been supported by SERC research grant GRG-54702

nodes are numbered last, A will have the bordered block diagonal form (BBDF) shown in eqn. 2, where subscript t indicates cut (or torn) nodes:

$$\begin{bmatrix} A_{11} & & & & A_{1t} \\ & A_{22} & & & A_{2t} \\ & & \ddots & & \vdots \\ & & & A_{mm} & A_{mt} \\ A_{t1} & A_{t2} & \cdots & A_{tm} & A_{tt} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \\ x_t \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \\ b_t \end{bmatrix} \quad (2)$$

The factorisation of m subnetworks (rows/columns from 1 to m) is mutually independent and can be performed by parallel processors. As all the communication between the subnetworks is via the last cutset block A_{tt} , the factorisation updates to the elements of this block can be added independently after all the processing of rows/columns 1 to m is finished. The F/B substitution follows a similar pattern of parallelly processed subnetworks and sequentially processed tearing nodes.

The efficiency of this standard parallel method depends on the size of the cutset block A_{tt} . If a large network is torn into a small number of subnetworks the size of the subnetworks is much greater than the size of the cutset block. This means that the amount of inter-processor communication is low when compared with the amount of uninterrupted computation on each processor (coarse grain parallelism). Also, the share of the sequential part of the algorithm (factorising and solving the cutset block) in the whole algorithm is small. The algorithm can be implemented efficiently on a MIMD architecture and good performance can be expected. On the other hand, if the network is torn into a greater number of subnetworks the granularity of parallelisation of the algorithm becomes finer, communication overheads grow and the share of the sequential part of the algorithm (e.g. processing of A_{tt}) increases.

The influence of the sequential part of any parallel algorithm on the expected speedup can be analysed using Amdahl's Law. The speedup S as a function of the number of parallel processing elements m can be expressed (in ideal conditions) as

$$S(m) = \frac{T}{t_s + \frac{t_p}{m}} \quad (3)$$

where T , t_s and t_p are the serial execution times of the entire code, of the nonparallel code portion, and of the parallel code portion, respectively. Dividing the numerator and denominator by T yields

$$S(m) = \frac{1}{\frac{t_s}{T} + \frac{t_p}{Tm}} = \frac{1}{\frac{1}{m} + W_s \left(1 - \frac{1}{m}\right)} \quad (4)$$

where $W_s = t_s/T$ is the share of the nonparallel part in the total sequential execution time. Fig. 1 contains graphs of $S(m)$ for several values of W_s . As little as 10% of the serial code portion (i.e. $W_s = 0.1$) restricts the expected speedup to a value of six no matter how many processors are used. Generally, the sequential part becomes more important and causes faster saturation as the number of processors is increased.

Factorisation of blocks from 1 to m introduces fill-ins in the last block A_{tt} . As the result the density of the nonzero elements in this block is usually much higher than that in the main subnetwork blocks, causing the computational complexity of its factorisation to grow

with a power of between two to three with the number of nodes in it. This means that the share of the sequential part of the code, represented by the processing of A_{tt} , increases steeply with the number of processors. The thicker line in Fig. 1 corresponds to an example analysed

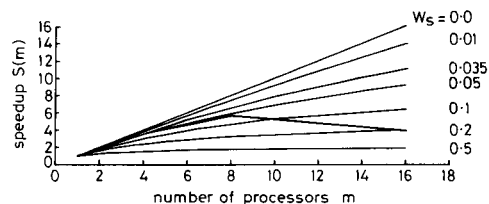


Fig. 1 Amdahl's law

in Reference 16, in which it was assumed that the share of A_{tt} in the factorisation is only 1% when using four processors. Increasing the number of processors to eight results in a modest increase in the speedup but further increase to 16 results in a rapid drop in the performance.

There are also other factors which contribute to the unimpressive performance of the standard clustering-based method. First, it is difficult to tear the network in such a way that the clusters obtained give a balanced computational load on each processor. Any imbalance deteriorates the performance of the algorithm even further. Secondly, the clustering itself influences the optimal node ordering and may introduce additional fill-ins slowing the algorithm even further [5, 7].

3 Use of factorisation tree for network clustering

The factorisation tree, or elimination tree, is one of the most important tools for understanding the factorisation-based solution of large, sparse linear equations. An excellent overview of the role of factorisation trees, although in relation to Cholesky factorisation, can be found in Reference 17.

The factorisation path for any node k , called the child node, is an ordered list of nodes starting at k . The list contains the index of the first nonzero element in column k of the lower triangular matrix L . This new node, the parent node, is taken as a column and the process is repeated until the last node is reached. Consider a simple ten node system shown in Fig. 2a. Solid lines correspond to the connections between the nodes, while dashed lines correspond to fill-ins. Each node is labelled with a number showing its position in minimum degree minimum length (MDML) ordering [18]. Fig. 2b shows the factorisation path graph or factorisation tree of the network. The meaning of numbers labelling the tree nodes is explained later.

As the factorisation path for each node determines a precedence relationship in the factorisation and substitution phases, it follows that nodes which do not belong to the same factorisation path can be eliminated in parallel. The same holds for F/B substitution. This concept has been used in References 13 and 14 to identify independent groups of nodes and to re-order the nodes in such a way that nodes within each group are numbered consecutively with remaining nodes being numbered last. As a result the A and LU matrices have a BBDF structure and the whole procedure can be seen as a refined clustering-based method. This fact has not been explicitly recognised in the quoted papers and it is analysed here in detail.

First, it is necessary to introduce some concepts from graph theory. A symmetric matrix A can be structurally represented by an undirected graph $G(A) = (X(A), E(A))$,

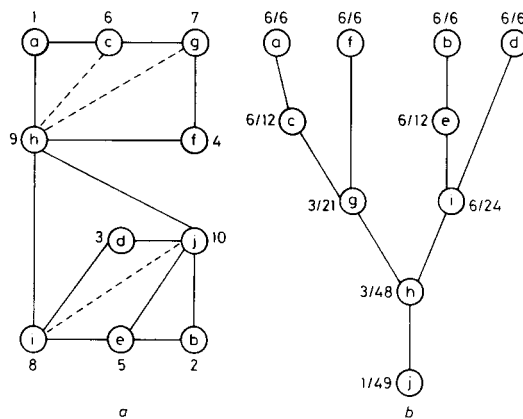


Fig. 2 Simple ten-node system
a Network diagram and fill-ins
b Factorisation tree with factorisation complexities/weights

where $X(A)$ is the set of nodes, $E(A)$ is the set of edges connecting the nodes. Nodes in $X(A)$ correspond to rows and columns of the matrix and edges in $E(A)$ correspond to nonzero off-diagonal entries. Hence, the network diagram with solid lines shown in Fig. 2a can be interpreted as the undirected graph $G(A)$. The filled graph $G(F)$ is the graph associated with filled matrix F (or LU) of A after all factorisation fill-ins have been introduced and corresponds to the full diagram of Fig. 2a (i.e. including the dashed lines). Obviously $G(A)$ is a subgraph of $G(F)$.

For any two nodes in the factorisation path, x_i and x_j , x_i is called the descendant of x_j if it has to be factorised before x_j . Node x_j is the ancestor of x_i . It also follows in this case that $i \leq j$. For example, node g is the ancestor of nodes a, c, f . Each node is the descendant and ancestor of itself. $T[x]$ is used to denote a subtree of the tree $T(A)$ rooted at the node x , which includes all the descendants of x in the tree T . For example, $T[g] = \{a, c, f, g\}$, $T[i] = \{b, e, d, i\}$. Two important theorems, quoted from Reference 17, are now introduced to help in understanding how factorisation tree partitioning works.

Theorem 1: States that for each node x_j , the subgraph of $G(A)$ $\{G(F)\}$ consisting of nodes in $T[x_j]$ is connected.

An interpretation of this theorem is that partitioning the tree into disjoint subtrees amounts to network clustering. Consider for example subtrees $T[g] = \{a, c, f, g\}$ and $T[i] = \{b, e, d, i\}$. Both subtrees correspond to clusters of interconnected nodes, as can be verified on Fig. 2a.

Theorem 2: Is more important and states that every topological ordering of the factorisation tree is an equivalent reordering of the given sparse matrix, where the topological ordering is such an order that numbers child nodes before their parent nodes.

Note that reordering the optimally ordered nodes in such a way that the matrix has BBDF structure, where each main block corresponds to one or more subtrees, is a topological reordering. This, according to Theorem 2, should introduce no extra fill-ins and the number of

arithmetic operations for factorisation should also be preserved. Fig. 3a shows the LU matrix of the network of Fig. 2 with the nodes ordered using MDML method,

	a	b	d	f	e	c	g	i	h	j		a	c	f	g	b	e	d	i	h	j
a	x						x					a	x	x							x
b		x										c	x	x	x						*
d			x									f		x	x						x
f				x								g	x	x	x						*
e					x							b				x	x				x
c						x						e				x	x	x			x
g							x					d									x
i								x				i							x	x	*
h									x			h	x	*	*	*	*	*	*	*	x
j										x		j	x	x	x	*	*	*	*	*	x

Fig. 3 (LU) matrix structure of example system

a MDML ordering
b Topological reordering giving BBDF structure
x original entry
+ fill-ins

while Fig. 3b corresponds to BBDF reordering with nodes on subtrees $T[g]$ and $T[i]$ grouped together. Both orderings are equivalent as far as the number of fill-ins and the complexity of solution is concerned.

The conclusion is that the factorisation-tree partitioning is a sparsity-preserving clustering of the network. This is important as generally a clustering method, by disturbing the optimal node ordering of a given network, may introduce additional fills and therefore slow down the algorithm [7]. Use of the factorisation tree prevents this possibility and consequently the parallel code, executed serially, should be equivalent to the best serial code. This hypothesis has been tested on a range of networks described in Section 8 and proved to be correct.

4 Balancing load on parallel processors

It is well known that equal loading on all processors is essential to ensure a success of any parallel method. The authors of References 13 and 14 attempted to balance the number of nodes that are assigned to each processor in the hope that this would balance the load on them. Another approach, first proposed in Reference 19 for Cholesky factorisation and adapted here to power system matrices, is based on an observation that the computational effort connected with factorisation and F/B substitution of a given node is not uniform for all the nodes. In fact, it is a function of the number of nonzero entries in a column of the L matrix. It can easily be verified that the computational complexity, defined as a number of multiplications-additions (mult-adds), of factorisation of a column with k nonzero entries is

$$C(k) = 1 + k + \frac{k(k+1)}{2} \quad (5)$$

For both forward and backward substitution the complexity is k . In the following analysis we concentrate on factorisation as it is more time consuming. It would appear that tree partitioning should be based on the complexity of factorisation of a given branch, rather than simply on the number of nodes in it. Define a weight of a subtree as the sum of factorisation complexities (given by eqn. 5) of all the nodes in it. Node labels in Fig. 2b correspond to factorisation complexities/weights.

The node-to-processor allocation algorithm can be based on balancing the weight of subtrees allocated to

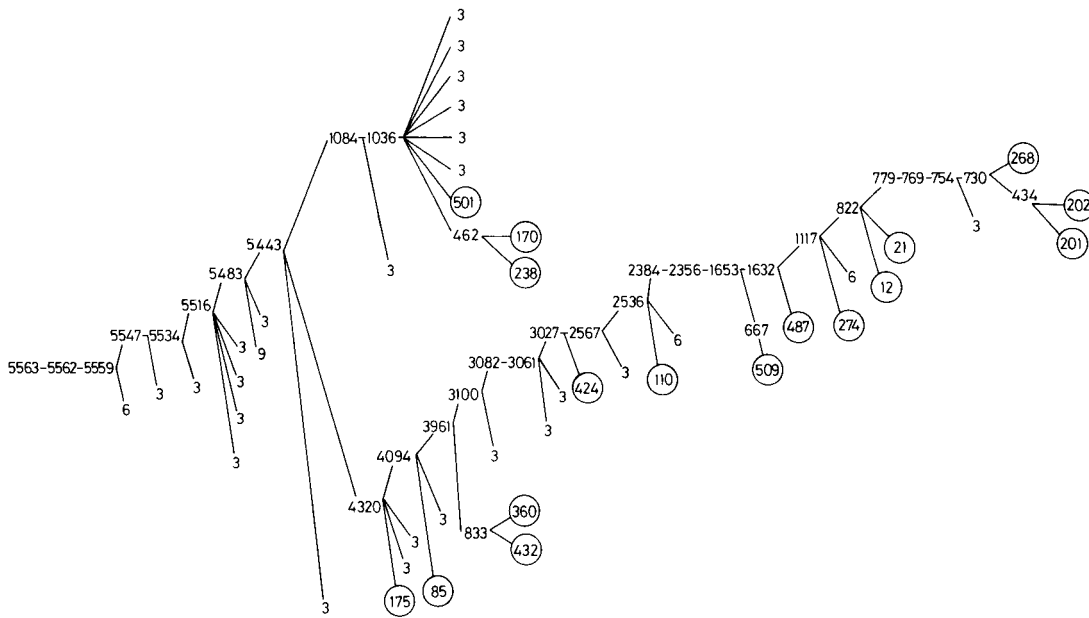


Fig. 6 Aggregated factorisation tree of CEGB 734 node system, MDML ordering

The minimum degree minimum length (MDML) algorithm [18] has been adopted as it provides a good compromise between the two objectives, but the algorithm has been modified to include a third tie-breaking strategy, applied when two or more nodes have the same degree and length. The criterion picks the node which has been least recently referenced. This requires time-stamping each uneliminated node when its neighbour in the filled graph is factorised and searching for the node with the smallest timestamp in the event of a tie. The time stamp may be just an iteration number. This ordering, referred to as minimum degree minimum length least recently used (MDMLLRU) [13], should discourage factorisation of nodes in one area only and encourage a balanced factorisation over the whole network. The extra time and complexity involved in implementing this third tie-break strategy is negligible.

The proposed method has proved successful, giving good results with well-balanced trees. For example, compare the factorisation trees of the CEGB 734 node system, MDML-ordered in Fig. 6 against MDMLLRU-ordered in Fig. 7 (the meaning of encircled groups of nodes is explained in the following Section). Both methods started from the same natural ordering. The height of the MDML tree is 35 with very unbalanced two main branches of the weights 1084 and 4320. The height of the MDMLLRU tree is 26 with two well-balanced main branches of the weight 2687 and 2839, respectively. The method has been tested statistically (for 500 random initial orderings) and has performed consistently better than the MDML method, giving smaller mean tree height (29.8 against 31.4) and shorter tallest tree encountered (34 against 46). The smallest tree was for both cases 23, which is understandable as MDMLLRU is a subset of MDML. A similar test for the IEEE 118 node network did not show any major differences between the orderings because the network was too small.

6 Recursively parallel method

The sequential part of the algorithm, i.e. processing of the tearing nodes, is the main factor limiting the parallel speedup. In this Section a new algorithm is proposed to exploit existing parallelism between tearing nodes at each

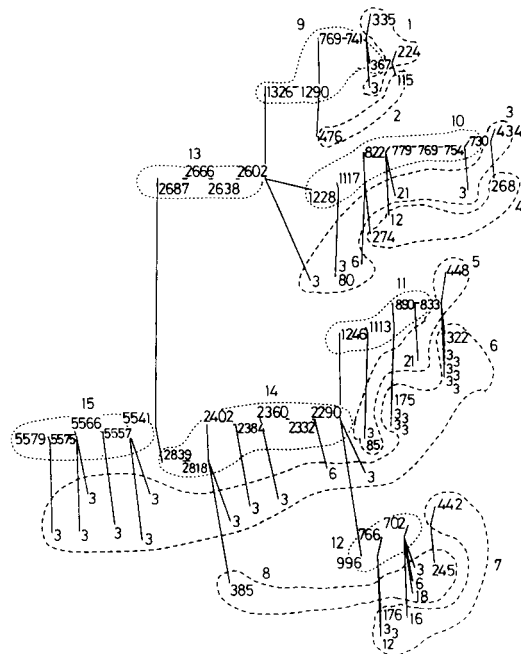


Fig. 7 Aggregated factorisation tree of CEGB 734 node system, MDMLLRU ordering

level of the factorisation tree. Consider again the factorisation tree of Fig. 2b and assume that there are four processors at our disposal. Using the standard method described in Section 2 we would have allocated nodes *a* and *c*, *b* and *e*, *d*, *f* to four consecutive processors and treated all other nodes as cut nodes to be processed serially by one of the processors with other processors idling. Processing in this way neglects a potential parallelism existing between nodes *g* and *i*. Instead of processing all the cut nodes using one processor we propose to process nodes *g* and *i* using two processors in parallel, and after that process *h* and *j* serially using one processor. We therefore intend to exploit existing parallelism between groups of cut nodes using idling processors.

The algorithm outlined can be best applied to a network partitioned in such a way that the resulting block factorisation tree has a shape of the binary tree. To explore this idea consider a general case, analysed in detail in Reference 16, of a network partitioned into 15 subnetworks to be processed by eight processors. Fig. 8a

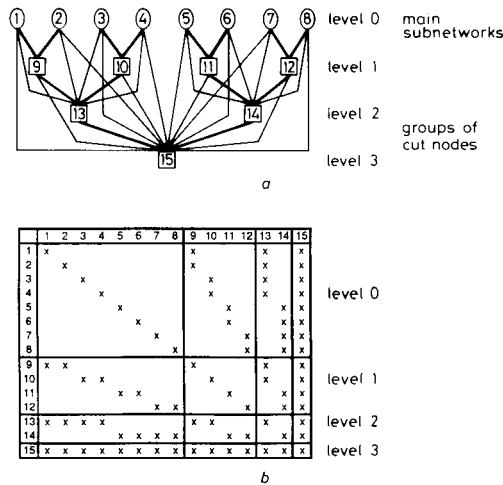


Fig. 8 General 15 subnetwork
 a Filled graph and the factorisation tree
 b RBBDF structure of *LU* matrix

shows the filled graph of the network with the edges corresponding to the allowed connections (or fill-ins) between the blocks. The resulting factorisation tree is indicated by the thicker lines. The structure of the *LU* matrix is shown in Fig. 8b. Nodes 1-8 represent main subnetworks while nodes 9-15 represent groups (subnetworks) of cut nodes. We propose that the factorisation of the network is executed in four stages corresponding to the four levels in the factorisation tree. In the first stage, the eight main clusters at level 0 are processed in parallel by eight processors. In the second stage, the four groups of cut nodes at level 1 are processed in parallel by four processors; stage 3 has two groups of cut nodes at level 2 processed by two processors, and finally the cut-node group 15 at level 3 is processed by one processor. Forward and backward substitution follows a similar pattern.

Theoretical analysis has shown [16] that the expected speedup can be improved from about 3.5 to approximately seven when using the method described. Obviously, not all the allowed connections between groups of nodes shown in Fig. 8a could be expected to exist in a

real power system. This is merely a general structure from which some elements may be omitted as required. However, it is useful as it allows the design of a general program capable of dealing with any particular case of this type.

The *LU* matrix has a characteristic structure, as the BDDF structure of level 0 and below is again repeated at levels 1 and below, and 2 and below. The matrix thus has recursive bordered block diagonal form (RBBDF) and the whole solution method is referred to as the recursively parallel (RP) method. Generally, the method results in a network being partitioned into *m* main subnetworks and (*m* - 1) groups (subnetworks) of tearing nodes. The network can then be solved by *m* parallel processors in $\log_2 m + 1$ sequential steps.

Fig. 7 shows application of the general scheme described to the aggregated factorisation tree of the CEGB 734 node network. Subtrees forming subnetworks have been circled together and given a number corresponding to subnetworks from Fig. 8. Factorisation complexities of the first eight main subnetworks vary between 550 (subnetwork 3) and 657 (subnetwork 8). The predicted factorisation speedup was 5.82, while the actual was 5.38. This represents a significant improvement with respect to the standard method speedup of 3.66. More results can be found in Table 1, Section 8.

When using the recursively parallel method the longest path in a tree forms the critical path of the algorithm which cannot be further parallelised. This, in connection with Amdal's law, allows a crude estimation of the optimal number of processors required. Assuming that factorisation complexity of all the nodes is roughly uniform, the number of nodes on the critical path (equal to the tree height) gives a measure of the sequential share of the algorithm (W_s from 1). For example, the IEEE 118 node network factorisation tree has a height of 12, which gives $W_s = 12/118 = 0.1$. Observing Fig. 1, one can estimate that the optimal number of processors for this system is eight with the expected speed-up being approximately four. Adding more processors results in a fast saturation of the gain. Similarly, for the CEGB 734 node network, the length of the critical path is 26, $W_s = 26/734 = 0.035$, and the expected speedup is about six when using eight processors and about ten when using 16 processors. The simulations confirmed the usefulness of this approximate method of prediction.

One of the main problems with any parallel processing method is minimising the amount of communication required between processors. For the recursively parallel method the traffic can be minimised by aggregating the messages as they are sent down the tree. For example, factorisation of subnetworks 1 and 2 at level 0, Fig. 8, produces updates to subnetworks 9, 13 and 15 at levels 1, 2 and 3. Similarly, factorisation of subnetworks 9 and 10 at level 1 produces updates to subnetworks 13 and 15, etc. These updates often refer to the same nodes in subnetworks at lower levels, and therefore may be aggregated (added) and sent down as a total rather than two separate components. In this way, the total amount of traffic can be significantly reduced. Another method of minimising the communication overheads is related to the task mapping and is considered in the following Section.

7 Task mapping onto processors

The placement of tasks onto processors is one of the most important parallel programming problems. If com-

putational tasks are placed onto processors in such a way that large amounts of interprocessor communication are required, all the benefits of parallel processing can quite easily be destroyed. For the recursively parallel method a task corresponds to processing one of the subnetworks from Fig. 8. It can be proved [20] that the best mapping strategy is to place two task from different levels of the factorisation tree on one processor. Fig. 9 shows an

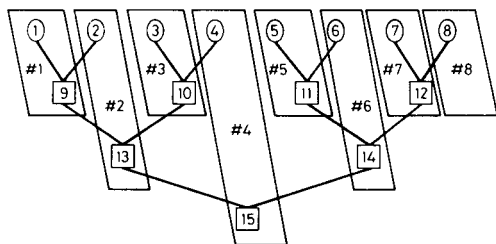


Fig. 9 Mapping of tasks to processors for 15 subnetwork system

○ main subnetworks
□ outset networks

example of the task mapping of the 15 subnetwork general case of Fig. 8. The strategy can be easily scaled up to a higher number of processors, say 32 or 64. This mapping strategy has additional benefit of minimising the communication overheads, as communication time between two different task placed on the same processors is negligible.

The most popular, off-the-shelf, multiprocessor systems are those using Immos Transputers. One limitation of the Transputer is that it has only four physical communication links available per processor making it difficult to implement interconnection topologies which require more than four connections to each processor. It can be shown [20] that the proposed task mapping procedure can be conveniently implemented on a planar processor topology requiring no more than four links per processor. This is scalable to any number of processors making it eminently suitable for use with the Transputer. Note that the proposed mapping strategy eliminates the need for global communication (i.e. all processors requiring a simultaneous access to the processor dealing with the tearing nodes) necessary for the standard parallel method and resulting in significant delays. In the recursively parallel method the communication is necessary only between the neighbouring processors in the tree structure of Fig. 9. Moreover, the messages are not transmitted all at the same time, but rather sequentially — first from the tree leaves down to the root (for factorisation and forward substitution) and then the other way around (for back substitution).

8 Simulation results

The recursively parallel method has been tested and compared against the standard parallel method on a number of networks including IEEE 118 node network and a 734 node representation of the CEGB network, using first a PC and then an array of Transputers. Zollenkopf's bifactorisation [4] method was used rather than more traditional *LU* factorisation.

Table 1 shows the simulation results for the two test systems performed on a PC and compares the recursively parallel (RP) method with the standard parallel network solution method described in Section 2, when the matrix is of BBDF structure and all cut nodes are processed serially. The speedup was calculated as the ratio of the execution time of the best serial algorithm to the parallel completion time (excluding communication overheads). The speedup is given separately for factorisation, F/B substitution and for the overall solution (i.e. factorisation + F/B substitution). In the factorisation phase the predicted speedup (calculated as described in Section 4) is compared with the actual speedup and the results show a good accuracy of prediction. Comparison between the proposed RP method and the standard BBDF method shows a significant improvement in the computational speedup.

Implementation of the RP method on an array of transputers has confirmed that the communication overheads do not present a significant obstacle. For example, the overall speedup (i.e. factorisation + F/B substitution) for the 734 node network was 3.6 when using an array of four transputers, which should be compared with 3.29 in the simulation. This apparent improvement is due to increased parallelism of computations made possible during parallel implementation, but difficult to predict or estimate in the simulation. The algorithm is still under development and further improvements similar to those suggested can be expected. Problems with implementing the PR method on a truly parallel system are not trivial and they will be the subject of another paper after a full investigation using more Transputers and a variety of networks is completed.

9 Conclusions

We have analysed a method for exploiting more of the existing parallelism in the solution of the algebraic network equations associated with power system problems. The proposed recursively parallel method, suitable for MIMD-processor architecture and based on the use of a factorisation tree, exploits not only parallelism existing between the main subnetworks, but also that existing between cut nodes. Additionally, a modified node-

Table 1: Simulation results for two test systems

System	No. of processors	Computational speedup						
		Factorisation			F/B substitution		Overall	
		RP predicted	RP actual	Standard BBDF	RP method	Standard BBDF	RP method	Standard BBDF
IEEE 118	2	1.55	1.55	1.55	1.62	1.62	1.497	1.497
	4	2.75	2.72	2.63	2.72	2.42	2.72	2.55
	8	4.02	4.41	3.28	3.86	2.94	4.17	3.13
CEGB 734	2	1.96	2.01	2.01	1.87	1.87	1.92	1.92
	4	3.56	3.34	3.14	3.08	2.95	3.29	3.12
	8	5.82	5.38	3.66	6.2	3.95	5.68	3.76
	16	7.5	6.67	3.91	8.14	4.95	7.1	4.2

ordering algorithm is proposed which gives short and well-balanced factorisation trees. A new method for network clustering based on estimation of the computational complexity is also proposed which does not deteriorate sparsity of the factor matrices, gives good balancing of the computational load between the parallel processors, allows visualisation of large networks, and accurately predicts the expected parallel speedup. A simulation of the proposed recursively parallel method has proved its effectiveness and it has been shown that the method suffers less from the speedup saturation effect of traditional parallel solutions.

10 References

- 1 CHAI, J.S., and BOSE, A.: 'Bottlenecks in parallel algorithms for power system stability analysis', *IEEE Trans.*, 1993, **PWRS-3**, pp. 9-15
- 2 TINNEY, W.F., and WALKER, J.W.: 'Direct solution of sparse network equations by optimally ordered triangular factorisation', *Proc. IEEE*, 1967, **55**, pp. 1801-1809
- 3 OGBUOBIRI, E.C., TINNEY, W.F., and WALKER, J.W.: 'Sparsity direct decomposition for Gaussian elimination on matrices', *IEEE Trans.*, 1970, **PAS-89**, (1), pp. 141-150
- 4 ZOLLENKOPF, K.: 'Bi-factorisation: basic computational algorithm and programming techniques', in 'Large sparse sets of linear equations' (Academic Press, 1971), pp. 75-96
- 5 WU, F.F.: 'Solution of large-scale networks by tearing', *IEEE Trans.*, 1976, **CAS-23**, (12), pp. 706-713
- 6 HATCHER, W., BRASCH, F., and VAN NESS, J.: 'A feasibility study for the solution of transient stability problems by multi-computer structures', *IEEE Trans.*, 1977, **PAS-96**, (6), pp. 1789-1797
- 7 ALVARADO, F.L., REITAN, D.K., and BAHARI-KASHANI, M.: 'Sparsity in diakoptic algorithms', *IEEE Trans.*, 1977, **PAS-96**, (5), pp. 1450-1459
- 8 UNDRILL, J.M., and HAPP, H.H.: 'Automatic sectionalisation of power system networks for network solution', *IEEE Trans.*, 1971, **PAS-90**, (1), pp. 46-52
- 9 SANGIOVANI-VINCENTELLI, A., CHEN, L., and CHAU, L.: 'An efficient heuristic algorithm for tearing large-scale networks', *IEEE Trans.*, 1977, **CAS-24**, (12), pp. 709-717
- 10 JESS, J.A., and KEES, G.H.: 'A data structure for parallel LU decomposition', *IEEE Trans.*, 1982, **C-31**, pp. 231-239
- 11 TINNEY, W.F., BRANDWAIN, V., and CHAN, S.M.: 'Sparse vector methods', *IEEE Trans.*, 1985, **PAS-104**, (2), pp. 295-301
- 12 PADILHA, A., and MORELATO, A.: 'A W-matrix methodology for solving sparse network equations on multiprocessor computers', *IEEE Trans.*, 1992, **PWRS-7**, (3), pp. 1023-1030
- 13 TAYLOR, A.J.E.: 'Techniques for power system simulation using multiple processors'. PhD thesis, University of Durham, 1990
- 14 LAU, K., TYLAVSKI, D.J., and BOSE, A.: 'Coarse grain scheduling in parallel triangular factorisation and solution of power system matrices', *IEEE Trans.*, 1991, **PWRS-6**, (2), pp. 708-714
- 15 BIALEK, J., and GREY, D.J.: 'An automatic clustering algorithm using factorisation tree for parallel power system simulation'. Proc. 7th Mediterranean Electrotechnical Conference MELECON'94, Antalya, Turkey, April, 1994, pp. 980-983
- 16 BIALEK, J.: 'Parallel solution of torn networks for power system simulation'. Proceeding of the 6th International Conference on *Present-day problems of power engineering*, Gliwice, Poland, 1993, vol. 2, pp. 75-82
- 17 LIU, J.W.H.: 'The role of elimination trees in sparse factorisation', *SIAM J. Matrix Anal. Appl.*, 1990, **11**, (1), pp. 134-172
- 18 BETANCOURT, R.: 'An efficient heuristic ordering algorithm for partial matrix refactorisation', *IEEE Trans.*, 1988, **PWRS-3**, (3), pp. 1181-1187
- 19 GEIST, G.A., and NG, E.: 'Task scheduling for parallel sparse Cholesky factorisation', *Int. J. Parallel Program.*, 1988, **18**, (4), pp. 291-313
- 20 GREY, D.J., and BIALEK, J.: 'A mutated tree architecture for real-time parallel power system simulation'. Proceedings of the 28 universities conference on *Power engineering*, Stafford, 1993, pp. 458-462