PARTITIONED SPARSE A⁻¹ METHODS

Fernando L. Alvarado, Senior Member
The University of Wisconsin - Madison

David C. Yu, Member
The University of Wisconsin - Milwaukee

Ramón Betancourt, Member
San Diego State University

Abstract

This paper solves the classic $Ax=b$ problem by constructing factored components of the inverses of L and U , the triangular factors of A . The number of additional fill-ins in the partitioned inverses of L and U can be made zero. The number of partitions is related to the path length of sparse vector methods. Allowing some fill-in in the partitioned inverses of L and U results in fewer partitions. Ordering algorithms most suitable for sparsity preservation in the inverses of L and U require additional fill-in in L and U themselves. Tests on practical power system matrices from 118 to 1993 nodes indicate that the proposed approach is competitive in serial environments, and appears more suitable for parallel environments. Because sparse vectors are not required, the approach works not only for short-circuit calculations but also for power flow and stability computations. **Keywords:** Power Flow, Stability, Parallel computing, Sparse Matrices, Linear equations, Partitioning.

Introduction

Sparse vector methods have changed the handling of sparse matrices in power systems. Sparse vector methods were introduced in detail by Tinney et. al [1], but are also described by Alvarado et. al. in [2, Section 5]. The fundamental idea within sparse vector methods is that many power system problems require repeat solutions to large sets of sparse linear equations of the form $Ax=b$, where the vector b is also sparse. The solution vector x is not sparse, but the user is often interested in a subset of the elements of x .

Sparse vector methods have found applicability where sparse right hand side vectors exist, such as the short circuit problem [2,3], the compensation problem [4,5], and, by modifying Newton's method, the power flow problem [6]. Sparse vector methods are related to the idea of recursive ordering of matrices for enhancing parallelism [7-11]. Also related is prior work on parallel inversion of sparse matrices [12], nested dissection [13-14], and the classic ordering schemes of Tinney [15].

Recent efforts have resulted in ordering techniques aimed at improving the performance of sparse vector methods [16,20] by reducing the average length of the factorization path during sparse vector solutions at a small penalty in added fills. Another ordering scheme that has the ability of improving the performance of sparse vector methods is Scheme W of Enns, Tinney and Alvarado [21].

Reference [21] laid the groundwork for the use of partitioned inverses of the factors of A . However, the method of using partitioned inverse factors was not deemed practical because of the fill-in in the W -matrices (the inverses of L and U). Although it was recognized that the fills could be reduced by partitioning W , the possibility of no additional fills in W was not seriously considered. This paper elaborates upon the idea of partitioned W -matrices, where either no fills or very few additional fills are created.

The methods in this paper are applicable to either serial or parallel environments, but there is a clear bias toward the latter. A detailed treatment of the application of the methods presented in this paper to parallel computer environments is beyond the scope of this paper. Reference [22], however, discusses this subject and presents results and observations from the application of some of these techniques to load flow solutions on a Cray X-MP/48 supercomputer.

Mathematical Background

Power systems problems often require the solution of sparse sets of linear equations of the form:

$$A \cdot x = b \quad (1)$$

The matrix A is first factored as $A = L \cdot D \cdot U$. The solution to this problem proceeds in three steps: forward substitution, diagonal scaling and back substitution. The computation of L and U is preceded by ordering of the rows and columns of A to minimize the number of new nonzero terms. Define:

$$W^L = L^{-1} \quad W^U = U^{-1} \quad (2)$$

W^L is lower triangular and sparse. The solution to the original problem can be expressed as:

$$x = W^U \cdot D^{-1} \cdot W^L \cdot b \quad (3)$$

Equation (3) may be solved in three steps

$$z = W^L \cdot b \quad y = D^{-1} \cdot z \quad x = W^U \cdot y \quad (4)$$

Within each step, all multiplications can be performed concurrently. The W matrices can be partitioned to enhance sparsity. The matrix L can be expressed as

$$L = L_1 \cdot L_2 \cdots L_n$$

where each L_i is an identity matrix except for the i -th column, which contains column i of L . Then

$$W^L = L^{-1} = L_n^{-1} \cdots L_2^{-1} \cdot L_1^{-1} = W_n^L \cdots W_2^L \cdot W_1^L$$

where W_i^L is L_i with its signs reversed on the off-diagonal elements. Using this expanded form of W in (3):

$$x = W_1^U \cdot W_2^U \cdots W_n^U \cdot D^{-1} \cdot W_n^L \cdots W_2^L \cdot W_1^L \cdot b \quad (5)$$

We can combine adjacent W matrices. If each W matrix is grouped into two partitions, the solution for x may be expressed as

$$x = W_a^U \cdot W_b^U \cdot D^{-1} \cdot W_b^L \cdot W_a^L \cdot b \quad (6)$$

Equation (6) is processed from right to left in five serial steps. All multiplications within each step are amenable to parallel processing.

Partitioned Factorization Paths

The factorization paths of a matrix were introduced with respect to sparse vector operations to expose precedence relations among operations on rows and columns of a matrix [1]. The factorization path for any node k is an ordered list of nodes starting at k . The list contains the number of the

89 SM 679-2 PWRs A paper recommended and approved by the IEEE Power System Engineering Committee of the IEEE Power Engineering Society for presentation at the IEEE/PES 1989 Summer Meeting, Long Beach, California, July 9 - 14, 1989. Manuscript submitted January 31, 1989; made available for printing June 9, 1989.

node of the first nonzero element in column k of L . This new node is then taken as a column and the process is repeated until the last node is reached.

The *length* of node i in the factorization path is defined as the maximum of all the path lengths from node i back to all its initial nodes (all the nodes that precede it in the path graph). Figure 1 illustrates the 20 node system from [1]. Figure 2 illustrates the corresponding matrix topology. Figure 3 illustrates the factorization path graph, with the *length* of every node indicated. Different orderings produce different numbers of fills and factorization path graphs. To minimize sparse vector operation counts it is desirable to decrease the length of the factorization paths.

The structure of W is implied by the path graph. For example, the nonzero entries in column i of W are given by the nodes that are included in the path of node i . Also, the nonzero entries in row i of W are given by the nodes in which path node i is included. References [12,18,19] give further details.

An approach to partition W without producing additional fills can be based on the path graph. If all the nodes at one level are numbered ahead of all the nodes at the next level, partitioning the matrix according to levels results in no fill in W . Thus, the number of partitions in W can be made equal to the maximum length in the graph. All the nodes at the same level would be in the same partition. No fills are created in this case since nodes within a partition will not be predecessors of each other. The concept of *length* can be made equivalent to the concept of *partition*.

Most ordering algorithms do not guarantee that all elements at the same level in the graph will be numbered consecutively. As a result, it is not possible with most orderings to partition the matrix into as many partitions as there are levels without at least some fill-in in W , unless the matrix is re-numbered.

Choosing elements from more than one level does not imply that fills in W will take place. If an element is chosen for a partition, such as node 18, it may be possible to choose another element from among its successors (node 19, for example) and still obtain no added fills, provided that all the neighbors of node 18 that have preceded it in the path are also neighbors of node 19. Thus, it is possible to have fewer partitions than the maximum path length and still have no additional fill-in in the partitioned W matrix.

Figure 1: A 20 node system from [1].

Figure 2: The topology of the L -matrix including fills. A partition according to PA2 that results in no fill-in in W is also shown.

Figure 3: Factorization path illustrating the concept of *length*. Node 20 is trivial and is not shown.

Once a matrix is available in LDU factored form, the number of multiplications required to perform a full forward substitution and back substitution is τ , where τ is the number of nonzeros in the matrix after all fills have been added. Somewhat surprisingly, the number of multiplications required to solve a problem using the partitioned factored inverse matrix is also τ , regardless of the number of partitions. Depending on the grouping, however, a few additional multiplications may be required to construct the inverse factors themselves, even though these have the same sparsity as L .

Partitioning after Ordering

The objective of partitioning is to reduce fill-in in W . We can inspect the pair-wise products of the elementary W_i matrices to see whether fill-in in W occurs. For the path graph and L -matrix shown in Figures 2 and 3, it can be seen that if a partition contains nodes 1 and 2, then the number of nonzeros in $W_2 \times W_1$ is the same as the number of non-zeros in $L_1 \times L_2$. However, if the partition contains rows 9 and 10, the number of nonzeros is $W_{10} \times W_9$ is five, while the number of non-zeros in $L_9 \times L_{10}$ is four; thus, fills result if these two nodes are in the same partition.

Based on this observation, algorithm (PA1) to partition a matrix and produce no fill-in in W is introduced. In this algorithm, consecutive nodes are grouped into the same partition without creating any fill-in in W as long as each node is from a different branch of the path graph. Define two arrays:

Generic ordering/partitioning algorithm:

- (1) Form valence classes for all nodes. The definition of "valence" varies depending on the desired algorithm. Typical valences are: degree, length, fills in L and fills in W. Initialize $i = 1$ and $p = 1$. Initialize all counts.
- (2) If i equals N , STOP. Otherwise
- (3) Select node with lowest valence as next pivot, provided its selection does not result in exceeding the number of nonzeros allowed in the current W partition. Break ties according to a "secondary criterion" and further ties according to "tertiary criterion."
- (4) If all nodes violate the number of nonzeros limit, then a new partition must be started. Re-initialize partition counts. For MLMD and MDML variations, reset length. For Gómez 1 and 3 variations, reset b vector. For Gómez 2 variations, reset m vector. Also reset fills in W counts. Select node with lowest valence as next pivot. Let $p = p + 1$.
- (5) Add fills to L . Add fills to W .
- (6) Update Valence Classes by re-computing the valences of any nodes whose valence may have changed.

This algorithm requires the use of certain procedures not detailed in the algorithm description. Steps 3 and 4 select a next pivot as the "lowest valence node." Valence can be defined in a variety of ways. If valence is defined as "degree" (the number of first neighbors of a node), then the primary algorithm is Scheme 2. However, if valence is defined as "length" then the primary algorithm is MLMD. If valence is defined as fills in W , the primary algorithm is Scheme W. Breaking ties is done using secondary and tertiary criteria. Many combinations are possible. For example:

- MLMD-P: Use minimum length as the primary criterion, minimum degree as a secondary criterion and fills in W as tertiary criterion. Nodes that result in exceeding the allowed number of fills are not eligible.
- Scheme W-P: Use minimum fills in W as the primary criterion, break ties according to minimum degree in L . Nodes that result in exceeding the allowed number of fills are not eligible for pivoting until the next partition is started.

Examples and Tests

We compare algorithms using five test systems ranging from 118 nodes to 1993 nodes. All examples correspond to actual power systems. For each of these systems, we compare several algorithmic combinations. The following ordering-only algorithms are considered:

- Tinney Schemes 2 and 3.
- Gómez and Franquelo algorithms 1, 2 and 3 (GF1, GF2 and GF3).
- Betancourt's MDML and MLMD algorithms.

For each of these algorithms we obtain:

- The number of elements in the factored L matrix.
- The number of partitions according to PA1 and PA2.
- The number of partitions according to PA3. Several amounts of fills per partition are permitted: 1%, 2%, 5%, 10% and 20%. For each case the tables indicate the actual number of partitions and the number of total fills in the partitioned W .

The following algorithms perform both ordering and partitioning:

- Partitioned variants of Gómez-Franquelo algorithms.

- A partitioned variant of MLMD (MLMD-P).
- A variant of Enns, Tinney and Alvarado's Scheme W.

For each algorithm, we illustrate the number of partitions and the number of fills in W for each test system. Results are summarized in Tables IV and V.

Ordering schemes based primarily on minimum degree result in more partitions than ordering schemes based primarily on other criteria. As long as there are no additional fills in W , partitioned sparse inverse methods can be used without computational penalty when solving full vector problems *even when a large number of partitions is required*. The only penalty of a large number of partitions is greater average path length when dealing with sparse vectors, and more serial steps in parallel environments.

Methods based on criteria other than minimum degree as the main criterion can result in fewer partitions at the expense of additional fills in L and U . It is always possible to find partitions that result in no additional fills in the W -matrices. However, giving up a few additional fills in W can reduce further the number of partitions.

Table IV: Partitions and fills in W -matrices after ordering. Top number is the number of partitions, bottom number (if any) is the number of fills in W .

(a) 118 node system, 179 original off-diagonal nonzeros

	Sch2	Sch3	GF1	GF2	GF3	MDML	MLMD
L nonzeros	265	269	265	264	285	265	303
PA1	57	64	48	19	19	50	12
PA2	51	57	41	13	12	43	6
PA3	22	24	12	6	7	17	6
1% (2)	31	38	29	10	8	29	5
PA3	16	18	12	6	7	13	6
2% (4)	48	52	33	18	15	37	13
PA3	9	12	8	5	5	8	5
5% (9)	69	84	51	37	32	57	20
PA3	6	8	6	4	4	6	4
10% (18)	88	115	79	54	51	80	53
PA3	5	5	4	3	3	4	3
20% (36)	135	160	110	99	80	108	79

(b) 707 node system, 969 original off-diagonal nonzeros

	Sch2	Sch3	GF1	GF2	GF3	MDML	MLMD
L nonzeros	1991	1973	1957	1936	2061	1908	2221
PA1	249	303	223	62	63	207	35
PA2	213	261	186	29	31	173	11
PA3	36	48	31	12	13	28	10
1% (10)	307	397	258	90	106	241	59
PA3	24	33	20	11	11	18	9
2% (19)	395	522	325	164	148	293	109
PA3	13	18	11	8	8	11	7
5% (48)	522	748	466	286	300	419	242
PA3	9	11	8	6	6	8	5
10% (97)	734	919	635	490	466	603	461
PA3	7	7	6	5	5	6	5
20% (194)	1098	1260	944	754	726	904	724

(c) 1993 node system, 3328 original off-diagonal nonzeros

	Sch2	Sch3	GF1	GF2	GF3	MDML	MLMD
L nonzeros	6075	5862	6108	6030	6502	6071	6871
PA1	683	821	600	220	138	582	43
PA2	563	711	487	127	71	470	15
PA3	43	54	36	20	14	33	11
1% (33)	1257	1623	1083	524	388	981	252
PA3	25	31	22	15	11	21	9
2% (67)	1462	1902	1329	787	631	1228	495
PA3	13	16	12	9	7	11	7
5% (166)	1998	2423	1749	1332	1084	1577	950
PA3	8	10	8	7	6	8	6
10% (333)	2484	3008	2350	2089	1649	2420	1647
PA3	7	7	6	5	5	6	5
20% (666)	3998	3944	3534	3081	2645	3307	2576

- f[i] is equal to the node that immediately follows node i in the path graph. This array is a concise and complete way of representing the entire path graph.
- b[i] represents the maximum node number among those nodes which precede node i in the path graph.

Algorithm PA1

- (1) Let $i = 1$ and $p = 1$
- (2) If i equals N , STOP (N is the final node in the path graph). Otherwise
- (3) Starting from node i , scan all consecutive nodes $i, i+1, i+2$, etc. until a node $i+m$ is found where $b(i+m) \neq 0$.
- (4) Group nodes i through $i+m-1$ identified in step (3).
- (5) Let the group identified in (4) become partition p . For every node j in the new partition p , find a node k such that $k = f(j)$. If $b(k)=j$, then let $b(k) = 0$.
- (6) Let $i = i+1$ and $p = p+1$, go to step (2).

The minimal number of partitions generated by PA1 is bound by the maximal level of the path graph. This can be illustrated by the 20-node example. The path graph shows that the maximum level of this example is 7. The last node in the matrix is a trivial node, since its L-matrix is the identity matrix and is not considered.

Using PA1, each node in the longest path of the graph appears in a different partition. However, there are eight partitions generated. The discrepancy is caused by node 9. Node 9 does not belong to the critical path, but cannot be grouped into either partition without causing some W-fills. When W-matrices are generated based on PA1, no additional multiplications are required *either* to generate the W-matrices *or* to perform the repeat solutions.

To reduce the number of partitions while still adding no fills, we introduce algorithm PA2. In this algorithm, more than one node from the same path branch is permitted into the partition, provided inclusion of the node does not result in W-fills. We use a function "CheckExistenceOf" which verifies whether nodes already exist whenever a potential fill in W is contemplated. If all potential fills in W already exist, it is safe to add the node to the partition.

Algorithm PA2

- (1) Let $i = 1$ and $p = 1$
- (2) If i equals N , STOP (N is the final node in the path graph). Otherwise
- (3) Starting from node i , scan all consecutive nodes $i, i+1, i+2$, etc. until a node $i+m$ is found where $b(i+m) \neq 0$.
- (4) Group nodes i through $i+m-1$ identified in step (3).
- (5) (a) Define set X as the set of nodes within group connected to node $i+m$.
- (b) If all nonzeros in lower column $i+m$ already exist in all lower columns of set X then add element to the group, go to step 2. Otherwise
- (c) Start a new partition. The current group becomes partition p . For every node j in the new partition p , find a node k such that $k = f(j)$. If $b(k)=j$, then let $b(k) = 0$. Continue to step (6).
- (6) Let $i = i+1$ and $p = p+1$, go to step (2).

Algorithm PA2 usually results in fewer partitions. Results for the 20-node example are illustrated in Table II. The number of partitions is reduced, particularly toward the lower right hand corner of the matrix. Although no fills result, generating the factored inverse can require a few additional multiplications and additions beyond those needed

to compute L and U . Once the factored inverse is generated, the solution steps require the same amount of computation as the usual forward and back substitution approach.

Allowing a few additional fills during the formation of W can reduce the number of partitions. The algorithm for partitioning while permitting some fills in W is PA3.

Algorithm PA3

- (1) Let $i = 1$ and $p = 1$. Also, reset the cumulative count for fills in W .
- (2) If i equals N , STOP (N is the final node in the path graph). Otherwise
- (3) Starting from node i , scan all consecutive nodes $i, i+1, i+2$, etc. until a node $i+m$ is found where $b(i+m) \neq 0$.
- (4) Group nodes i through $i+m-1$ identified in step (3).
- (5) (a) Define set X as the set of nodes within group connected to node $i+m$.
- (b) For every nonzero in lower column $i+m$ not in a lower column of set X increase W -fill count.
- (c) If the cumulative number of W -fills is less than the pre-set allowed number, add element to group. Update the structure of the W -matrix to include these fills and go to step 2.
- (d) If the cumulative number of W -fills exceeds the pre-set number, then a new partition must be started. The node group becomes partition p . For every node j in the new partition p , find a node k such that $k = f(j)$. If $b(k)=j$, then let $b(k) = 0$. Re-initialize the W -fill count. Continue to step (6).
- (6) Let $i = i+1$ and $p = p+1$, go to step (2).

This algorithm differs from PA2 only in step (5), where instead of simply verifying whether fills in W occur, the number of W fills is counted, and the structure of the W matrix is updated. This algorithm requires keeping track of both the structure of LDU and the structure of W .

The results of allowing up to 3 fills per W partition for the 20-node system are illustrated in Table III. The fills that result if W is not partitioned is 42.

Table I: Partitions according to PA1

Nodes	1 - 8	9	10 11	12 - 14	15 16	17	18	19
W-fills	0	0	0	0	0	0	0	0

Table II: Partitions according to PA2.

Nodes	1 - 8	9	10 11	12 - 14	15 16	17-19
W-fills	0	0	0	0	0	0

Table III: Partitions according to PA3.

Nodes	1 - 9	10 - 14	15 - 19
W-fills	1	3	2

Combined Partitioning and Ordering

The number of partitions can play a dominant role in parallel environments. Partitioning schemes based on a-priori ordering of the matrix are likely to be less effective (although much simpler) than algorithms for matrix partitioning that consider both ordering and partitioning. We can modify those ordering algorithms (such as MLMD) that consider criteria other than minimum degree to reset the criteria every time a new partition is started. Algorithms for combined ordering and partitioning are based on existing ordering algorithms, modified to also keep track of fills in W . The following generic algorithm is the basic combined ordering and partitioning algorithm.

Table V: Partitions/fills using combined algorithms. Top number is partitions, bottom number is fills.

(a) 118 node system.

	GF1-P	GF2-P	GF3-P	MLMD-P	SchW-P
0%	10/0	10/0	14/0	6/0	6/0
1% (2)	7/12	6/11	7/13	5/6	6/10
2% (4)	6/26	5/20	6/27	5/11	5/16
5% (9)	5/46	4/35	5/38	4/36	4/35
10% (18)	3/72	3/58	3/55	4/52	4/58
20% (36)	3/170	2/96	2/87	3/85	3/88

(b) 707 node system.

	GF1-P	GF2-P	GF3-P	MLMD-P	SchW-P
0%	26/0	26/0	27/0	9/0	9/0
1% (2)	13/117	13/104	14/130	8/61	9/60
2% (4)	10/193	10/167	10/191	7/111	8/126
5% (9)	7/337	7/313	7/317	6/247	7/268
10% (18)	5/567	5/492	5/528	5/434	6/446
20% (36)	4/910	4/756	3/717	4/717	5/734

(c) 1993 node system.

	GF1-P	GF2-P	GF3-P	MLMD-P	SchW-P
0%	107/0	107/0	78/0	12/0	12/0
1% (33)	28/857	18/466	18/562	9/240	9/284
2% (67)	19/1214	13/843	11/748	8/455	8/514
5% (166)	10/1712	8/1284	7/1265	6/849	7/1064
10% (333)	7/2537	6/1950	5/1766	5/1490	6/1700
20% (666)	5/3223	4/3167	3/2608	4/2644	5/2652

We illustrate some topology maps. These maps shed light into the nature of ordering and partitioning algorithms on the sparsity pattern of the matrix. Figure 4 illustrates the general effect of ordering algorithms on the sparsity pattern. Figure 5 illustrates the effect of matrix size on one specific combination of ordering algorithm (MLMD) and partitioning algorithm (PA3). Figure 6 illustrates the effect of various partitioning and ordering algorithm combinations. It is ironic that sparsity, which started with banded matrices, now comes to the conclusion that what we prefer is an empty blocked-band near the diagonal.

Figures 7 and 8 illustrate the effect of algorithm and of system size on the number of partitions. Some algorithm combinations result in few partitions even when no fills are allowed. Some algorithms are relatively insensitive to system size, others are not.

Conclusions

It is possible to have zero additional fills in partitioned sparse inverse algorithms at a surprisingly low computational expense. When no additional fills are produced, the number of partitions can be quite low. Using simple partitioning after ordering algorithms, some of the algorithms can result in zero fill-in with as little as six to fifteen partitions. If 20% fill-in is permitted, the number of partitions can be as low as 3. Algorithms that combine ordering and partitioning are more complex but generally give the best results in terms of number of partitions. They also perform relatively well as far as amount of fill-in.

When applying the proposed algorithms to repeat solutions in serial environments, the speeds attainable are comparable to those of conventional methods, except for a few additional fills in L and U. However, the proposed methods (particularly those that allow some additional fill-in in W) are able to reduce the number of partitions in W substantially. In a parallel environment, fewer partitions represents fewer serial steps. Therefore, the proposed methods should prove advantageous in concurrent computational environments, whether the right hand side vectors are sparse or full. This makes the method applicable to stability and power flow calculations.

Acknowledgement

We thank Mr. Brian Johnson for helping produce the sparse matrix topology maps.

(a) Ordering according to Tinney Scheme 2

(b) Ordering according to Gómez-Franquelo 2.

(c) Ordering according to Betancourt's MLMD.

Figure 4: Effect of the ordering algorithm on the sparsity pattern of a 707 node system.

(a) A 352 node system.

(a) MLMD followed by PA2.

(b) A 1084 node system.

(b) MLMD followed by PA3, 10% fills allowed per partition.

(c) A 1993 node system

Figure 5: The effect of matrix size on sparsity pattern and the partition structure of a matrix, using MLMD and PA2.

(c) SchW-P, 10% fills allowed per partition.

Figure 6: A comparison of the effect of partition algorithms on the sparsity structure and the partition pattern for the 707 node system.

Figure 7: Partitions vs. fills per partition, 1993 node system.

Figure 8: Partitions vs. size with PA2 (no fills in W).

References

- [1] W. F. Tinney, V. Brandwajn and S. M. Chan, "Sparse Vector Methods," IEEE Trans. on Power App. and Systems, Vol. 104, No. 2, Feb. 1985, pp. 295-301.
- [2] F. L. Alvarado, S. K. Mong and M. K. Enns, "A Fault Program with Macros, Monitors and Direct Compensation in Mutual Groups," IEEE Trans. on Power App. and Systems, May 1985, pp. 1109-1120.
- [3] K. Takahashi, J. Fagan and M. Chen, "Formation of a Sparse Bus Impedance Matrix and Its Application in Short Circuit Studies," 1973 PICA Conference Proc., Minneapolis, Minnesota, June 1973, pp. 63-71.
- [4] O. Alsac, B. Stott and W. F. Tinney, "Sparsity Oriented Compensation Methods for Modified Network Solutions," IEEE Trans. on Power App. and Systems, Vol. PAS-102, No. 5, pp. 1050-1060, May 1983.
- [5] S. M. Chan and V. Brandwajn, "Partial Matrix Refactorization," IEEE Trans. on Power App. and Systems, Vol. PWR-1, No. 1, pp. 193-200, Feb. 1986.
- [6] R. Bacher and W. F. Tinney, "Faster Local Power Flow Solutions: The Zero Mismatch Approach," IEEE 1989 Winter Power Meeting, New York, Feb. 1989.
- [7] H. S. Stone, "An Efficient Parallel Algorithm for the Solution of a Tridiagonal System of Equations," J. of the ACM, Vol. 20, No. 1, Jan. 1973, pp. 183-208.
- [8] D. A. Calahan, "Parallel Solution of Sparse Simultaneous Linear Equations," Proc. 11th Allerton Conf. on Circuit and System Theory, U. of Illinois, 1973.
- [9] F. L. Alvarado, "Parallel Solution of Transient Problems by Trapezoidal Integration," IEEE Trans. Power App. & Syst., May-June 1979, pp. 1080-1090.
- [10] F. L. Alvarado and S. Y. Lee, "Sparsity Preservation in Global Numerical Formulations of Ordinary Differential Equations," International Conf. on Electric Power Problems: The Mathematical Challenge (SIAM), Seattle, Wash. March 18-20, 1980.
- [11] A. Abur, "A Parallel Scheme for the Forward/Backward Substitutions in Solving Linear Equations," IEEE Trans. in Power Systems, Nov. 1988, pp. 1471-1478.
- [12] R. Betancourt and F. L. Alvarado, "Parallel Inversion of Sparse Matrices," IEEE Trans. on Energy Systems, Feb. 1986, pp. 74-81.
- [13] A. George "Nested Dissection of a Regular Finite Element Mesh," SIAM J. Num. Anal, 1973, Vol. 10, pp. 345-363.
- [14] I. S. Duff, A. M. Erisman and J. K. Reid, "On George's Nested Dissection Method," SIAM J. of Numerical Analysis, 1976, Vol. 10, pp. 686-695.
- [15] W. F. Tinney and J. W. Walker, "Direct Solutions of Sparse Network Equations by Optimally Ordered Triangular Factorization," Proc. of IEEE, Vol. 55, No. 11, Nov. 1967, pp. 1801-1809.
- [16] R. Betancourt, "Efficient Parallel Processing Technique for Inverting Matrices with Random Sparsity," IEE Proc. Vol. 133, Part E, July 1986, pp. 235-240.
- [17] A. Gómez and L. G. Franquelo, "Node Ordering Algorithms for Sparse Vector Method Improvement," IEEE Trans. in Power Systems, Feb. 1988, pp. 73-79.
- [18] R. Betancourt, "An Efficient Heuristic Ordering Algorithm for Partial Matrix Refactorization," IEEE Trans. on Power Systems, Aug. 1988, pp. 1181-1187.
- [19] R. Betancourt, closure to [18] above.
- [20] A. Gómez and L. G. Franquelo, "An Efficient Ordering Algorithm to Improve Sparse Vector Methods," IEEE Trans. in Power Systems, Nov. 1988, pp. 1538-1544.
- [21] M. K. Enns, W. F. Tinney and F. L. Alvarado, "Sparse Matrix Inverse Factors," Paper 88 SM 728-8, 1988 Summer Power Meeting, July 19-24, 1988, Portland, Oregon.
- [22] A. Gómez and R. Betancourt, "Implementation of the Fast Decoupled Load Flow on Vector Computers," to be submitted to IEEE 1990 Winter Power Meeting.

Fernando L. Alvarado was born in Lima, Peru in 1945. He received the BEE and PE degrees from the National University of Engineering in Lima, Peru, the MS degree from Clarkson College (now Clarkson University) in Potsdam, New York, and the Ph.D. degree from the University of Michigan in 1972. He joined the University of Toledo in 1972. Since 1975 he has been with the University of Wisconsin in Madison, where he is currently a Professor of Electrical and Computer Engineering.

David C. Yu was born in Kaohsiung, Taiwan, R. O. C. on December 12, 1955. he received the B. S. degree in electrical engineering from the Chung-Yuan University in Taiwan, and the M.S. and Ph.D. degrees in electrical engineering from the University of Oklahoma in 1977, 1979 and 1983 respectively. Currently, he is an assistant professor at the University of Wisconsin - Milwaukee.

Ramón Betancourt was born in Tijuana, Mexico in 1953. He received his B.S. degree in Engineering from the University of Guadalajara in Mexico. He received an M.S. degree in Electrical Engineering in 1981, a second M.S. degree in Computer Science in 1984, and a Ph.D degree in electrical engineering in 1984, all from the University of Wisconsin-Madison. In 1984 he joined San Diego State University as an Assistant Professor. He is currently an Associate Professor at San Diego State University.

Discussion

B.F. WOLLENBERG, Control Data Corporation, Plymouth MN: The use of parallel processors to solve power system network analysis problems is obviously going to require development of new techniques to take advantage of computer architectures. Methods of solving sparse linear equations with parallel processors are at the leading edge of such developments.

In this paper the authors have addressed the ability to efficiently partition and order the factor inverse matrices to take advantage of parallel processors. However, given that much work in power system analysis still must be done on current serial architectures, do the techniques reported in this paper give us an advantage in using serial machines?

Specifically, how does the author's technique compare, as far as operations count, on a simple $Ax=b$ solution that is ordered by Tinney scheme 2? How does it compare, again as far as operations count, to repeat solutions programmed by state of the art sparse vector methods and ordered by Tinney Scheme 2?

Manuscript received August 1, 1989.

The authors thank Dr. Wollenberg for his interest in this paper. This paper has described a new methodology for dealing with sparse matrices. The methodology includes: (1) ordering techniques, (2) partitioning techniques, (3) the construction of the W-matrices and (4) the use of W-matrices instead of ordinary repeat solutions. Comparisons can consider many variants of these four aspects of the proposed methodology. The main contributions of this paper have been item (2) and the recognition of the advantages of the overall approach in parallel environments.

Dr. Wollenberg wants to know how the proposed methodology performs in *serial* environments as far as repeat solutions are concerned. The answer depends on the intended use of the method and on the exact method combination adopted. For the proposed approach, let us assume the use of MLMD for ordering and either PA1, PA2 or PA3 (10%) for partitioning. Certainly, any of the other orderings proposed in the paper could have been studied. For the "conventional" method, we use both scheme 2 (the normal case) and MLMD. We compare only multiplication counts.

Consider first full right hand side vectors. In a strictly serial situation, the amount of computation as measured by the number of multiplications for a full repeat solution is slightly greater for the proposed method in either of its three partition options, mainly because of the large number of fills induced by the MLMD ordering algorithm (more on MLMD ordering later). Results for two test systems are illustrated in Table C-I.

TABLE C-I
Multiplications for full repeat solution

System	118	352
Conventional Scheme 2	648	1570
Conventional MLMD	704	1776
MLMD PA1	704	1776
MLMD PA2	704	1776
MLMD PA3 10%	886	2312
(max W-fills/partition)	(47)	(119)

We consider singleton sparse vectors next. We assume that only elements of the solution vector that correspond to nonzeros in the original right hand side vector are of interest. The average number of multiplications required to perform a sparse forward and sparse back substitution for two test systems is illustrated in Table C-II.

TABLE C-II
Average Multiplications, sparse repeat solution, singleton RHS

System	118	352
Conventional Scheme 2	71.4	99.1
Conventional MLMD	58.2	77.6
MLMD PA1	58.2	77.6
MLMD PA2	51.6	70.9
MLMD PA3 10% (fills)	46.5	54.8

The exact numbers in these tables depend on tie-breaking and exact details of implementation. However, some observations hold true in these and many other simulations. For example, the PA3 ordering algorithm (and to a lesser extent PA2) gives lower average operation counts than PA1, and usually lower than conventional scheme 2. These lower operation counts cannot be explained as a result of the shorter paths. The explanation is that the proposed approach using W-matrices does not need to access the entire path for the repeat solution process, only a subset of the path. In effect, the additional operations that are required during the construction of the W-matrix (as described in the paper body) result in a reduction of the number of operations required for the repeat solution step.

In the course of the studies performed for this closure, we have come up with several observations regarding MLMD ordering:

- We have constructed artificial networks where MLMD as originally described and implemented performs poorly, while scheme 2 performs much better. This occurs because, if MLMD considers only length as the primary criterion, it is possible for it to choose a node with a large ordinary valence that will produce a large amount of fill-in. A slight modification to MLMD to take into account this possibility performs considerably better while retaining all other desirable properties.
- Shorter path lengths do not automatically translate into fewer multiplications during sparse repeat solutions.
- The performance of most ordering algorithms (including scheme 2) is affected by the initial ordering of the network nodes. However, MLMD appears to be more sensitive to initial node numbering than scheme 2. A related observation is that the MLMD results as presented in [18] are subject to greater variability than anticipated. This does not seem to be the case for MDML. Cases exist where an unmodified MLMD ordering can result in longer average factorization path than scheme 2. MLMD should not be used without some modification.

In view of Dr. Wollenberg's comment that much work must still be done on serial architectures, we suggest that strictly serial architectures are nowadays the exception, not the norm. Most high performance computers use as a minimum a certain degree of "pipelining" for all operations. The proposed methodology takes care of speed gains attainable by pipelining in an automatic fashion, without the need for explicit user intervention or specialized coding. Thus, the proposed methodologies should perform better than the figures above suggest, even in seemingly serial environments.

The main contributions of this paper have been the partitioning technique and the study of the potential advantages of the methodology in parallel environments. The answers to Dr. Wollenberg's specific questions are: For full repeat solutions in *strictly* serial machines, the proposed method can be worse. For sparse repeat solutions in *strictly* serial machines the proposed method (particularly PA3 using the *modified* version of MLMD) can be better.

Manuscript received September 1, 1989.