ALGORITHMS FOR EFFICIENT VECTORIZATION OF REPEATED SPARSE POWER SYSTEM NETWORK COMPUTATIONS

Cevdet Aykanat, *Member* Özlem Özgü
Computer Engineering Department
Bilkent University
Ankara, Turkey

Nezih Güven, *Member*
Electrical Engineering Department
Middle East Technical University
Ankara, Turkey

Abstract – **Standard sparsity-based algorithms used in power system applications need to be restructured for efficient vectorization due to the extremely short vectors processed. Further, intrinsic architectural features of vector computers such as chaining and sectioning should also be exploited for utmost performance. This paper presents novel data storage schemes and vectorization algorithms that resolve the recurrence problem, exploit chaining and minimize the number of indirect element selections in the repeated solution of sparse linear system of equations widely encountered in various power system problems. The proposed schemes are also applied and experimented for the vectorization of power mismatch calculations arising in the solution phase of FDLF which involves typical repeated sparse power network computations. The relative performances of the proposed and existing vectorization schemes are evaluated, both theoretically and experimentally on IBM 3090/VF.**

1. INTRODUCTION

The solution of randomly sparse linear system of algebraic equations is one of the most challenging problems for vectorization and parallelization. Many compute-intensive and time-critical power system problems such as load-flow, contingency analysis, state estimation, transient stability, optimal power flow, etc., require the solution of this form of equations. Therefore, speeding up the solution of such equations by exploiting the state-of-the-art computer architectures is a crucial topic in power systems [1]. Problems of this form are most effectively solved in two phases; triangular factorization phase and Forward/Backward Substitution (FBS) phase. The vectorization and parallelization of each individual phase have been the topic of many recent research efforts [2–11].

In vector/parallel processing technology, it is a well known fact that the best sequential algorithms may not lead to the best vector/parallel algorithms. The existing algorithms should be restructured or new algorithms need to be developed for utmost efficiency on vector/parallel computers. In this context, standard sparsity based algorithms used in power system network computations need to be restructured for efficient vectorization due to the extremely short vectors processed. W-matrix formulation is a good example for restructuring the conventional methods for the sake efficient vectorization and parallelization of FBS computations [2, 4, 6]. Similarly, power system applications which involve structural network changes require special attention since efficient vectorization is only possible with the use of static data structures. Bus type changes enforced by Q-limit check in *Fast Decoupled Load Flow* (FDLF) [12], can be considered as a typical example to such applications. Approaches which model such changes as structural modifications in B'' and perform complete refactorization of the modified B'' are not suitable for vectorization since such structural modifications in B'' necessitate re-forming the reactive W matrix. However, there exist efficient formulations which enable the use of the original (in value) triangular factors and hence the same reactive W-matrix for the solution of modified reactive load flow equations by avoiding complete and partial refactorization [11, 13]. Furthermore, formulations which model Q-limit enforcement as non-structural modifications to B'' and perform partial refactorization [14] can also be exploited in vectorization since they do not disturb the structure of the reactive W matrix. In contingency analysis, post compensation can be effectively exploited in order to utilize the original real and reactive W matrices to account for the structural changes corresponding to branch outages [11, 13].

In this work, we propose efficient data storage schemes and vectorization algorithms for the repeated FBS computations. The solution phase of FDLF, which involves typical sparse power network computations, is used as the benchmark for the proposed algorithms. The proposed data storage schemes are exploited to develop efficient vectorization algorithms for the repeated real/reactive mismatch computations which constitute the most time consuming part of the solution phase of FDLF. Data storage schemes and vectorization algorithms proposed in this paper resolve the recurrence problem, exploit chaining and minimize the number of indirect element selections to attain utmost vector performance. This paper also provides a general overview of the improvements that can be expected by means of vector processing and of the guidelines that must be followed to achieve efficient vectorization of power system problems.

2. OVERVIEW OF VECTOR PROCESSING

Vector processing achieves improvement in system throughput by exploiting *pipelining*. To achieve pipelining, an operation is divided into a sequence of subtasks, each of which is executed by a specialized hardware stage that operates concurrently with other stages in the pipeline. Successive tasks are streamed into the pipe and executed in an overlapped fashion at the subtask level. In FORTRAN, pipelining can be exploited during the execution of DO-loops. Vectorizing compilers convert each vectorizable DO-loop into a loop consisting of vector instructions. Each vector instruction is associated with a *start-up time* overhead which corresponds to the time required for the initiation of the vector instruction execution, plus the time needed to fill the pipeline. Hence, optimizing an application for a vector computer involves arranging the data structures and the algorithm in a way to produce long vectorizable DO-loops.

Vectors processed during the execution of a vectorizable DO-loop may be of any length that will fit in storage. However, each vector computer is identified with a *section-size* K

which denotes the length of the vector registers in that computer (e.g., K= $64, 128, 256$). Vectors of length greater than K are sectioned, and only K elements are processed at a time, except for the last section which may be shorter than K. Vectorizing compilers generate a sectioning loop for each vectorizable DO-loop. Hence, each section is associated with an overall start-up time overhead which is equal to the sum of the start-up time overheads of the individual vector instructions in the loop.

Vector computers provide the chaining facility to further improve the performance of pipelining. *Chaining* allows the execution of two successive vector instructions to be overlapped where vector elements produced by as the result of one instruction pipeline are passed on-the-fly to a subsequent instruction pipeline which needs them as operand elements. In vector computers, advantages of instruction chaining are obtained by providing several of the most important combinations of operations with single compound vector instructions, such as Multiply-Add instruction. When both multiplication and addition pipelines become full, one result of the compound operation will be delivered per machine cycle. The following DO-loop illustrates the chaining of multiplication with addition:

```
DO j = jstart, jend
    BV(j) = BV(j) + V(IX(j)) × WV(j)        (1)
ENDDO
```

Vector computers load, store or process vectors in storage in one of two ways: by sequential addressing (contiguously or with stride), or by indirect element selection. Indirect element selection, or *gather-scatter*, permits vector elements to be loaded, stored or processed directly in an arbitrary sequence. In indirect addressing, the memory locations of the vector elements to be accessed are indicated by a vector of integer indices, which must be previously stored in a vector register. In DO-loop (1), vectors WV, BV and IX are accessed sequentially, whereas vector V is accessed indirectly with addresses specified by the IX vector. The performance of vector computers degrades drastically during indirect vector accesses. Hence, the number of indirect vector accesses should be minimized for efficient vectorization.

Unfortunately, vectorizing compilers generate scalar code for the following type of DO-loops:

```
DO j = jstart, jend
    BV(IX(j)) = BV(IX(j)) + WV(j)        (2)
ENDDO
```

This DO-loop contains apparent dependence due to indexing of the BV array by the IX array in both sides of the statement in (2). There can be a *recurrence* if two elements of the IX array have the same value. These recurrences make the result of one j iteration to be dependent on the results of the previous ones and hence scalar execution is mandatory to obtain correct results. Since such DO-loops are widely encountered during the vectorization of sparse power network computations, the recurrence problem is a crucial bottleneck for efficient vectorization. The DO-loop (2) can be executed in vector mode by enforcing the compiler to vectorize this DO-loop through the use of ignore-dependence type directives. However, a scheme should be developed to prevent the incorrect results that can occur due to recurrences.

3. FORWARD/BACKWARD SUBSTITUTION

The FBS phase in the solution of linear system of equations $Ax = b$, with an $N \times N$ coefficient matrix factorized in LDL^t form, consists of the following steps:

$$(a)\;\; L\,z = b; \quad (b)\;\; D\,y = z; \quad (c)\;\; L^t\,x = y. \tag{3}$$

Diagonal Scaling (DS) step (3.b) is suitable for vectorization since it can be formulated as the multiplication of two dense vectors (of sizes N) by storing the reciprocals of the diagonal elements. The loops of Forward Substitution (FS) step (3.a) and Backward Substitution (BS) step (3.c) can be vectorized on a vector computer with hardware support for scatter/gather operations. Unfortunately, in power system applications, these vectorized inner loops yield considerably poor performance since average vector length is very short.

Instead of performing the conventional FS and BS elimination processes, solution of $A\,x = b$ can be computed as

$$(a)\;\; z = W\,b; \quad (b)\;\; y = D^{-1}\,z; \quad (c)\;\; x = W^t\,y. \tag{4}$$

Here, $W = L^{-1}$ is called the inverse-factor. The advantage of (4) over (3) is that inherently sequential FS and BS computations are replaced by sparse matrix-vector products. However, experimental results show that the inverse-factor W may have many more non-zero entries compared with the factor L. Partitioning is proposed to reduce the W matrix fill-ins [4].

In partitioning schemes, the factor L is expressed as $L = L_1 \ldots L_N$, where the elemental factor matrix L_i is an identity matrix except for the i-th column which contains the corresponding column of L. Thus, $W = W_N \ldots W_1$ where the elemental inverse-factor matrix $W_i = L_i^{-1}$ is simply L_i with the negated off-diagonal entries. Consider gathering successive elemental L_i matrices into $L_{p_1}, L_{p_2}, \ldots L_{p_k}$ so that $W = W_{p_k} \cdots W_{p_2} W_{p_1}$. Hence, Eq. (4) is transformed into:

$$(a)\; z = W_{p_k} \cdots W_{p_1} b; \; (b)\; y = D^{-1} z; \; (c)\; x = W_{p_1}^t \cdots W_{p_k}^t y. \tag{5}$$

Various algorithms have been proposed for W–matrix partitioning which produce zero or only a prefixed maximum number of fill-ins [4, 6]. The simplest algorithm that produces no fill-ins exploits the *Factorization Path Graph* (FPG) concept. In this scheme, nodes at the same level of FPG are gathered into the same partition so that the number of partitions is equal to the depth of the FPG. Various ordering algorithms such as MD-MNP, MD-ML, ML-MD, $\cdots$, etc., have also been proposed to reduce the total number of levels in the resulting FPG [15, 16].

Data storage schemes for the off-diagonal non-zero elements in the W–partition matrices determine the structure of the vectorization algorithm to be used in the FBS phase. According to the number of vectors maintained for each partition, data storage schemes can be broadly classified as: (i) Single Vector Per Partition (SVPP), and (ii) Multiple Vectors Per Partition (MVPP) schemes. SVPP (MVPP) methods treat the non-zero elements of each partition as a single (multiple) vector(s) for a particular operation in each partition. MVPP methods introduce more start-up time overhead than SVPP methods. Nevertheless, MVPP methods can be exploited to reduce the number of recurrences and indirections as will be explained later.

3.1. Single Vector per Partition Methods

In this data storage scheme, non-zero elements of W-partition matrices are stored (column-wise), in partition order, in WV vector together with their row and column indices in RIX and CIX arrays, respectively. The partition pointer array PP contains pointers to the beginning indices of W-partition matrices in WV, RIX and CIX. The schemes proposed by Gomez and Betancourt [5] and Granelli et al. [8] utilize this data storage scheme which is illustrated in Fig. 2(a) for the second level of the L factor of the B' matrix for the IEEE-14 network in Fig. 1. Columns of L given in Fig. 1 are permuted in level order. Since each partition is taken as one level of the FPG, Fig. 1 illustrates the sparsity structure of W-partition matrices as well.

The FS phase of the approach proposed by Gomez and Betancourt [5] involves the following two DO-loops for each partition i:

```
DO j = PP(i), PP(i + 1) − 1
    WVR(j) = WV(j) × BV(CIX(j))                    (6.a)
ENDDO
DO j = PP(i), PP(i + 1) − 1
    BV(RIX(j)) = BV(RIX(j)) + WVR(j)               (6.b)
ENDDO
```

Here, WVR, of size M, denotes a real working array which is used to keep the multiplication results and M denotes the total number of off-diagonal non-zero elements in the W-partition matrices. The real array BV, of size N, is the right hand side vector (b in 5.a) on which the solution (z in 5.a) is rewritten. DO-loops (6.a) and (6.b) perform the multiplication and addition operations involved in each sparse matrix-vector product in (5.a), respectively. The DO-loop structure of the BS phase can easily be obtained by interchanging CIX with RIX in (6). In the FS(BS) phase, the addition DO-loop (6.b) is not vectorized by the compiler because of the possible recurrent indices in the RIX(CIX) array. Hence, only multiplications involved in the FBS phase are vectorized in this scheme, which will be referred to as GB hereafter.

The scheme proposed by Granelli et al. [8] is an improvement to scheme GB to vectorize the addition operations. In this scheme, recurrence-free row and column index vectors RIXRF and CIXRF are generated by replacing all *partition-basis* recurrences in RIX and CIX vectors, respectively, by $N+1$. The partition-basis recurrent row indices replaced by $N+1$'s in RIX are stored in RRIX together with their location indices in RRIXIX. Pointers to the beginning indices of partition-basis recurrence sets in RRIX and RRIXIX are stored in RRPP. The recurrences in the CIX array are maintained by similar integer arrays RCIX, RCIXIX and RCPP. This data storage scheme is illustrated in Fig. 2(b) for the second level of the L factor of the B' matrix in Fig. 1. Using this storage scheme, the implementation of Granelli's method for the FS phase can be obtained by replacing the addition DO-loop (6.b) by the following two DO-loops.

```
DO j = PP(i), PP(i + 1) − 1
    BV(RIXRF(j)) = BV(RIXRF(j)) + WVR(j)                 (7.a)
ENDDO
DO r = RRPP(i), RRPP(i + 1) − 1
    BV(RRIX(r)) = BV(RRIX(r)) + WVR(RRIXIX(r))           (7.b)
ENDDO
```

The DO-loop structure of the BS phase is similar. This scheme will be referred to as GR1 hereafter. Note that, N+1 is the only partition-wise recurrent index in RIXRF and CIXRF arrays. This ensures that all incorrect addition results with recurrent row indices will only contaminate BV(N+1). Thus, the compiler can safely be enforced to vectorize DO-loop (7.a). However, after a particular execution of this DO-loop, the addition phase of the corresponding partition is not completed since multiplication results corresponding to the recurrent row indices have not yet been considered for addition. These results are processed for addition in the scalar DO-loop (7.b).

Although scheme GR1 is a successful attempt to vectorize the addition operations, it does not exploit chaining since the multiplication and addition operations are vectorized in two different DO-loops. Chaining in this application can only be exploited by combining the multiplication and addition DO-loops into a single vectorizable DO-loop. However, this requires a new solution to the recurrence problem. In the following section, we propose an efficient scheme to resolve the recurrence problem which also enables chaining.

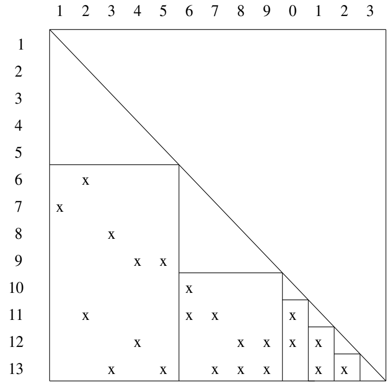

Figure 1: The sparsity structure of the factor and W-partition matrices of the B' matrix for the IEEE-14 network.

(a)

		10	11	12	13	14	15	16	17	
CIX	...	6	6	7	7	8	8	9	9	...
RIX	...	10	11	11	13	12	13	12	13	...

(b)

		10	11	12	13	14	15	16	17	
RIXRF	...	10	11	14	13	12	14	14	14	...

	1	2	3	4	5	6	
RRIX	9	13	11	13	12	13	...
RRIXIX	8	9	12	15	16	17	...

(c)

		10	11	12	13	14	15	16	17	
RIXRF	...	10	11	16	13	12	17	18	19	...

	1	2	3	4	5	6	
RRIX	9	13	11	13	12	13	...

Figure 2: The single-vector per partition data storage schemes for the FS phase: (a) GB, (b) GR1, (c) Proposed PR1.

The Proposed SVPP Scheme (PR1)

In scheme GR1, all multiplication results are saved in a temporary array WVR so that multiplication results corresponding to the recurrent elements can be selected from this array for scalar additions in a later step. However, the use of WVR should be avoided to achieve chaining. In the absence of WVR, multiplication results corresponding to the recurrent elements should be stored in the extended BV locations, BV($N+1$), BV($N+2$), ..., BV($N+R$), for scalar additions in a later step. Here, R denotes the total number of recurrences in the RIX and CIX arrays.

In the proposed scheme PR1, partition-wise recurrence-free row (RIXRF) and column (CIXRF) index vectors are constructed in a different manner. Each recurrence in the RIX (CIX) array is replaced with N+r in the RIXRF (CIXRF) array where r denotes the index of the next available recurrence location in the extended BV array. The partition-wise recurrence-free index arrays RIXRF, CIXRF and recurrence arrays RRIX, RRPP, RCIX and RCPP can easily be constructed, in linear time. Figure 2(c) illustrates the proposed data storage scheme for the FS phase of the W-partition matrices given in Fig. 1. The proposed scheme avoids the use of WVR, RRIXIX and RCIXIX arrays required in the GR1 scheme. In this scheme, chaining in the FS phase is achieved by the following DO-loops for each partition i:

```
DO j = PP(i), PP(i+1) − 1
    BV((RIXRF(j)) = BV(RIXRF(j)) + WV(j) × BV(CIX(j))    (8.a)
ENDDO
DO r = RRPP(i), RRPP(i + 1) − 1
    BV((RRIX(r)) = BV(RRIX(r)) + BV(N + r)               (8.b)
ENDDO
```

The DO-loop structure of the BS phase is similar. The DO-loop (8.a) achieves the chaining of addition and multiplication operations. Due to chaining in this DO-loop, correct multiplication results corresponding to the recurrent elements are added, on-the-fly, to the appropriate extended BV locations. Hence, extended BV locations should contain zeroes at the beginning of computations. This initialization loop is a vectorizable DO-loop with relatively long vector length equal to R.

The compound DO-loop (8.a) contains two types of apparent dependencies. The first is through indexing of the BV array by the RIXRF vector in both sides of (8.a). This dependence does not constitute any problem since RIXRF is a partition-wise recurrence-free array. The second type is through the use of the indices of the RIXRF and CIX arrays as pointers to the elements of the BV array in opposite sides of (8.a). Fortunately, all row indices associated with non-zero elements in each level are strictly greater than all column indices associated with those elements. That is, there is no level-basis recurrence between RIXRF and CIX arrays. Hence, the latter type of recurrences can be avoided by adopting *level-wise* partitioning. Consequently, the compiler can safely be enforced to vectorize DO-loop (8.a) to achieve chaining.

In partitioned scheme W, it is not mandatory for elements in a partition to be picked from the same level in the FPG. Nevertheless, adopting level-wise partitioning prevents cross recurrences between RIXRF and CIX (CIXRF and RIX) during the FS (BS) phase in DO-loop (8.a), and hence, substantially reduces the total number of scalar additions. In general, initial levels of the FPG already consist of long vectors enabling efficient vectorization. On the contrary, levels towards the bottom of the tree contain short vectors with large recurrence ratios. Hence, the relative advantages of GR1 and PR1 over GB decline in those levels. In this work, we gather those last levels into a single *multi-level last partition*. This last partition concept is also discussed for efficient parallelization in [10]. Adopting multi-level last partition enables a considerably long vector but results in a substantially large number of recurrences. Therefore, in the last partition, we have chosen to utilize scheme GB which vectorizes only the multiplication operations and avoids redundant addition operations. The last partition approach is adopted in all FBS vectorization schemes discussed in this paper.

The proposed scheme PR1 achieves substantial performance improvement in vectorization over scheme GR1 through chaining. For example, on IBM 3090/VF, PR1 reduces the number of delivery cycles by 18% and start-up time overhead by 25%. Chaining achieves this performance increase by avoiding the store and load operations for multiplication results. In the scalar DO-loop (8.b) of the proposed scheme, extended locations of the BV array are accessed in an orderly fashion for processing recurrent elements. However, in the scalar DO-loop (7.b) of GR1 scheme, WVR array is accessed indirectly with addresses specified by the elements of the RRIXIX array. Thus, the scalar performance of the proposed scheme is also expected to be slightly better than that of GR1 scheme in processing the recurrent elements.

Intra- / Inter- Section Recurrences

Consider a multi-section level with $s > 1$ sections. The vector facility creates a sectioning loop which iterates s times to vectorize DO-loop (8.a). In different iterations of the sectioning loop, elements belonging to different sections of RIXRF (CIXRF) will be used as address pointers to access the elements of the BV array. So, recurrences in RIX and CIX arrays can be classified as *inter-section* and *intra-section*. Inter-section recurrences are the recurrences between different sections whereas intra-section recurrences are the recurrences within the same section. Inter-section recurrences do not have any potential to yield incorrect results since they are processed in different iterations of the sectioning loop. Hence, only intra-section recurrences should be considered while generating the RIXRF and CIXRF arrays.

Here, we propose an efficient *round-robin* re-ordering algorithm which exploits this intra-section recurrence concept to minimize the number of redundant scalar operations. The proposed algorithm collects (in linear time) the non-zero elements with the same row (column) indices in a level and scatters them to the successive sections of that level in a modular sequence for the FS (BS) phase. During this re-ordering process, i-th appearances of a recurrent row (column) index in different sections of the RIXRF (CIXRF) array are replaced by the same extended BV location index $N{+}r{+}i{-}1$ for $i > 1$. Note that, first appearances of a recurrent index in different sections remain unchanged. The number of extended BV location assignments for a recurrent index determines the number of redundant scalar addition operations associated with that index. Hence, this scheme reduces the number of scalar additions required for a recurrent index ix with recurrence degree d_{ix} from $d_{ix} - 1$ of PR1 scheme to $\lceil d_{ix}/s \rceil - 1$ in a level with s sections. The proposed algorithm concurrently constructs the arrays required to maintain unavoidable recurrences during the re-ordering process. Note that, both W and W^t partition matrices are stored in this scheme.

3.2 Multiple Vector Per Partition Methods

In these data storage schemes, each sparse W–partition matrix is compressed into a relatively dense matrix. This compression is such that the off-diagonal non-zero elements of the W–partition matrices are allocated to contiguous locations of the columns of the compressed matrices. The number of columns in the compressed matrices are much less than those of the original ones.

Granelli et al. introduced the *pseudo–column* concept, or shortly *pscol*, in generating the compressed W matrices [11]. The main objective behind their *pscol* scheme, referred here as GR2, is to avoid the recurrence problem totally. In scheme GR2, the elements of an individual partition matrix whose row (column) indices appear for the i–th time are temporarily stored in the i–th *pscol* of a scratch compressed matrix for the FS (BS) phase. Fig. 3 illustrates this scheme for the first two levels of the matrix in Fig. 1 where "x","c" and "z" denote the non-zero elements compressed into the first, second and third *pscol*'s, respectively. Partition matrices condensed in this manner may contain *pscol*'s with intervening zeros as is the case for the first W-partition of Fig. 3. Scheme GR2 further compresses each *pscol* in order to avoid the processing of intervening zero entries. Compressed *pscol*'s of W and W^t partition matrices are stored in two different vectors together with their row and column indices.

Scheme GR2 executes only one multiplication DO-loop, similar to DO-loop (6.a), for each partition in both FS and BS phases. However, it executes one addition DO-loop, similar to DO-loop (6.b), for each *pscol* of the partition. Thus, GR2 scheme can be considered as a hybrid scheme. In this scheme, the addition DO-loops can be safely enforced for vectorization since the *pscol*'s of W and W^t partition matrices are already recurrence free. Hence, this scheme totally avoids the redundant scalar addition operations.

The Proposed MVPP Schemes (PR2–4)

In this section, we propose three MVPP schemes. The first one, PR2, incorporates chaining into Granelli's *pscol* scheme,

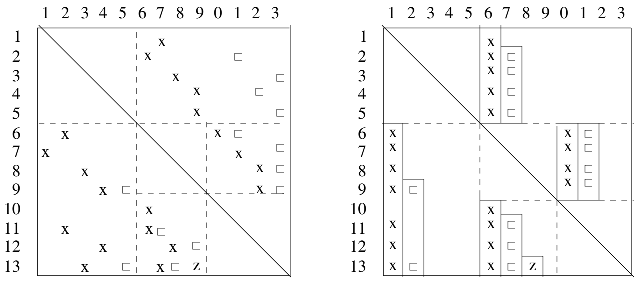

Figure 3: Granelli's pseudo–column data storage scheme for the first two levels of the B' matrix given in Fig. 1.

GR2. Multiplication and addition DO-loops for each *pscol* can be safely chained in both FS and BS phases by adopting level-wise partitioning as mentioned earlier for PR1. Note that, the proposed PR2 scheme is truly a MVPP approach.

The aim behind MVPP approach utilized in GR2 and PR2 schemes is to avoid redundant scalar additions. However, these MVPP schemes still require extensive use of indirections thru both row and column indexing as in the SVPP approaches. The other proposed schemes PR3 and PR4 aim at minimizing the number of indirections besides avoiding redundant scalar additions and exploiting chaining. Recall that row indices of successive entries in each column of a dense matrix are consecutive. However, *pscol*'s in scheme GR2 do not carry this property. In proposed schemes PR3 and PR4, rows of W and W^t partition matrices are sorted (in linear time) in ascending order. Then, i–th off-diagonal non-zero element in each row of a particular partition matrix is stored into the i–th *pscol* of that partition. Due to sorted row ordering, these *pscol*'s do not contain any intervening zeros thus avoiding the need for further compression. Furthermore, the row indices of the non-zero elements in each *pscol* appear in sequence. Hence, we call pseudo–columns obtained in this manner as *pseudo-dense–column*s, or *psdcol* shortly. Fig. 4 illustrates the proposed *psdcol* scheme for the first two levels of the matrix in Fig. 1 where "x" and "c" denote the non-zero elements compressed into the first and second *psdcol*'s, respectively.

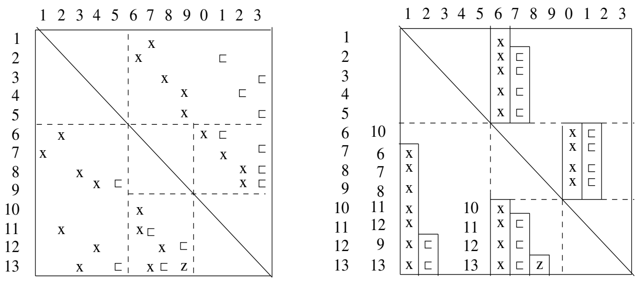

Figure 4: The proposed pseudo-dense–column data storage scheme for the first two levels of the B' matrix given in Fig. 1.

In the proposed data storage scheme, *psdcol*'s of W and W^t partition matrices are stored consecutively, in partition order, in two different vectors WVFS and WVBS together with their column indices in CIXFS and CIXBS vectors, respectively. Two pointer arrays PSCFS and PSCBS contain beginning indices of successive *psdcol*'s. It is sufficient to store only the row indices of the first elements of successive *psdcol*'s in RPFS and RPBS vectors since the row indices in each *psdcol* are successive. New vectorization schemes PR3 and PR4 are developed based on this *psdcol* concept. Since the vectorization of the BS phase is the same in both schemes, it is discussed first and discussion of FS phase is deferred. The BS phase is vectorized by executing the following DO-loop for each *psdcol* i:

$$\begin{array}{l} \mathrm{DO}\ j = \mathrm{PSCBS}(i),\ \mathrm{PSCBS}(i+1)-1 \\ \quad \mathrm{BV}(j-\Delta)\ =\ \mathrm{BV}(j-\Delta) + \mathrm{WVBS}(j) \times \mathrm{BV}(\mathrm{CIXBS}(j)) \\ \mathrm{ENDDO} \end{array} \tag{9}$$

Here, $\Delta\ = \mathrm{PSCBS}(i) - \mathrm{RPBS}(i)$ denotes the constant offset between the indices of the elements in WVBS array and their respective row indices. That is, index $j-\Delta$ denotes the row index of WVBS(j). This offset enables the sequential load/store of the BV array elements for update. DO-loop (9) shows that the proposed PR3 and PR4 schemes totally avoid the recurrence problem in the BS phase as in PR2 by adopting level-wise partitioning. Moreover, these two schemes reduce the total number of indirect element selections from $3m$ of PR2 to m in a partition with m elements. Hence, PR3-4 is surely the most efficient MVPP scheme for the BS phase.

Note that, the original row indices of different W^t partition matrices are disjoint. Unfortunately, this is not true for the W partition matrices. Hence, different row orderings among different W partition matrices complicate the vectorization for the FS phase. The original row indices of the permuted non-zero rows of W partition matrices are stored, in partition order, in the row permutation array RPM. Hence, each block of indices in RPM holds the original row indices of the successive non-zero elements in the first *psdcol* of each partition. Similarly, successive blocks of a real scratch vector SB are used to compute the results of the successive partition matrices. A partition pointer array RPMPP contains pointers to the beginning indices of partition blocks in RPM and SB vectors.

Appropriate entries of the BV vector are gathered into the scratch SB vector for update just before starting the sparse matrix-vector product for each partition. The non-zero elements of the first *psdcol*'s of each partition make contributions to all BV vector entries needed and updated in the respective partition matrix-vector product. In scheme PR3, we incorporate this gather operation into the update computations by executing the following DO-loop for the first *psdcol* f of partition p:

$$\begin{array}{l} \mathrm{DO}\ j = \mathrm{PSCFS}(f),\ \mathrm{PSCFS}(f+1)-1 \\ \quad \mathrm{SB}(j-\Delta 1)\ =\ \mathrm{BV}(\mathrm{RPM}(j-\Delta 1)) + \\ \qquad\qquad \mathrm{WVFS}(j) \times \mathrm{BV}(\mathrm{CIXFS}(j)) \\ \mathrm{ENDDO} \end{array} \tag{10}$$

where the constant offset $\Delta 1 =\ \mathrm{PSCFS}(f) - \mathrm{RPMPP}(p)$. Then, the contributions of the remaining *psdcol*'s can be computed and added into the scratch SB vector by executing the following DO-loop for each *psdcol* i that remains in partition p:

$$\begin{array}{l} \mathrm{DO}\ j = \mathrm{PSCFS}(i),\ \mathrm{PSCFS}(i+1)-1 \\ \quad \mathrm{SB}(j-\Delta 2)\ =\ \mathrm{SB}(j-\Delta 2) + \mathrm{WVFS}(j) \times \mathrm{BV}(\mathrm{CIXFS}(j)) \\ \mathrm{ENDDO} \end{array} \tag{11}$$

where the constant offset $\Delta 2\ =\ \mathrm{PSCFS}(i+1)\ -\ \mathrm{RPMPP}(p+1)$. Finally, results in the scratch SB vector are scattered into the appropriate locations of the BV vector as follows:

$$\begin{array}{l} \mathrm{DO}\ j = \mathrm{RPMPP}(p),\ \mathrm{RPMPP}(p+1)-1 \\ \quad \mathrm{BV}(\mathrm{RPM}(j)) = \mathrm{SB}(j) \\ \mathrm{ENDDO} \end{array} \tag{12}$$

As seen in DO-loops (10-12), PR3 reduces the number of indirect vector accesses compared to the FS phase of PR2 while achieving recurrence-free DO-loops with chaining. Consider a partition with d *psdcol*'s, and m non-zero elements such that $m\ =\ \sum_{i=1}^{d} n_i$ where n_i denotes the number of non-zero elements in the i-th *psdcol* of that partition. Since the lengths of DO-loops (10) and (12) are both n_1, PR3 scheme drastically reduces the number of indirect element selections from $3m$ of PR2 to $m + 2n_1$ in the FS phase. However, a careful analysis reveals the fact that sequential stores/loads to/from the SB vector in (10)/(12) are redundant compared to PR2. Hence, scheme PR3 can be considered as introducing $2n_1$ redundant sequential load/stores for the sake of efficient vectorization.

Here, we propose another scheme PR4 which avoids these redundant operations. Consider the execution of DO-loop (11) for a particular *psdcol* i, $1<i<d$, of a partition with d *psdcol*'s. The first $n_i - n_{i+1}$ updates stored into SB correspond to the final update results of that partition, because $n_i \geq n_{i+1}$. Thus, these updates can be immediately scattered to the BV vector by indexing thru RPM vector, avoiding the redundant sequential stores to SB. The rest n_{i+1} entries of the i-th *psdcol* can be handled by a second DO-loop similar to (11). DO-loop (10) for the first *psdcol* can similarly be decomposed into two DO-loops. Since updates caused by the last *psdcol* are final updates, only one DO-loop is sufficient. The PR4 scheme achieves the same number of indirections as PR3 while eliminating the redundant load/stores in PR3 and thus resulting in no redundancy like PR2. The only drawback of PR4 over PR3 is the increase in the total start-up time overhead due to the execution of two DO-loops for each *psdcol* except the last ones.

4. POWER MISMATCH COMPUTATIONS

This section presents the application of the proposed data storage schemes and algorithms for the vectorization of repeated power mismatch computations encountered in the solution phase of FDLF. The following formulation is adopted here for computing the right hand side vectors of the FDLF equations:

$$\Delta P_k/V_k \;=\; P_k^s/V_k \;-\; G_{kk}V_k \;-\; \sum_{h\neq k} Y_{kh}V_h\cos\alpha_{kh} \qquad (13)$$

$$\Delta Q_k/V_k \;=\; Q_k^s/V_k \;+\; B_{kk}V_k \;-\; \sum_{h\neq k} Y_{kh}V_h\sin\alpha_{kh} \qquad (14)$$

where $\alpha_{kh} = \Theta_k - \Theta_h - \delta_{kh}$. In this formulation diagonal and off-diagonal non-zero elements of the Y_B matrix are stored in rectangular $(G_{kk} + \jmath B_{kk})$ and polar $(Y_{kh}\ e^{\jmath\delta_{kh}})$ forms, respectively. Here, V_k and Θ_k denote the magnitude and phase angle of bus k voltage. In the right hand sides of Eqs. (13) and (14), the second and third terms denote the normalized contributions to the real/reactive powers of bus k due to the diagonal and off-diagonal non-zero elements of the Y_B matrix, referred here as *diagonal contributions* and *off-diagonal contributions*, respectively. By this formulation, normalized bus mismatch powers are computed directly instead of computing bus powers first and then the respective normalized mismatch powers.

Power mismatch computations in FDLF utilize the non-zero elements of Y_B. The diagonal contribution computations can be efficiently vectorized by performing operations on dense vectors. However, the data storage scheme used for storing the off-diagonal non-zero entries of the Y_B matrix is a crucial factor in the vectorization process. These schemes can also be broadly classified as *Single Vector* and *Multiple Vector* schemes. In the scheme proposed by Gomez and Betancourt [5], non-zero elements in the upper half of the symmetric Y_B matrix are stored as a single vector. In the two schemes proposed by Granelli et al. [8, 11], the off-diagonal non-zero elements of the Y_B are stored as multiple consecutive vectors such that each vector corresponds to a *pscol* of the Y_B matrix. One of these two schemes exploit the symmetry of the Y_B matrix whereas the other does not. These three schemes utilize the single-vector approach for the vectorization of the contribution computations. In Gomez's scheme, the addition of computed power contributions to appropriate entries of a real/reactive power vector is performed by a single efficient scalar DO-loop based on loop unrolling. Granelli et al. exploit their *pscol* concept to vectorize these addition operations by avoiding redundant scalar operations. Hence, Gomez's scheme is a single-vector scheme whereas Granelli's schemes can be considered as hybrid schemes. Unfortunately, all these schemes require two auxiliary integer vectors to store the row and column indices of the respective Y_B elements, thus necessitating extensive use of indirect addressing.

In this work, we propose a truly multiple-vector scheme based on our *psdcol* concept. Our scheme aims at minimizing the number of indirect vector accesses while achieving the vectorized addition of contributions to mismatch vectors. The proposed scheme exploits chaining whenever possible.

Permuting the buses in ascending degree order prevents intervening zeros in the resulting *psdcol*'s. Since Eq. (14) is computed only for the PQ buses whereas Eq. (13) is computed for all buses, computation of Eq. (14) will necessitate extra indirection overhead to locate the entries of the PQ buses in the *psdcol*'s. The proposed solution is to order Y_B such that all PQ buses are permuted in ascending degree order before the PV buses are permuted in descending degree order. Then, i-th off-diagonal non-zero elements of successive rows constitute the i-th *psdcol*. Fig. 5 shows the resulting structure after applying the proposed ordering on Y_B of IEEE-14 network. Note that, the proposed scheme does not exploit the symmetry of Y_B.

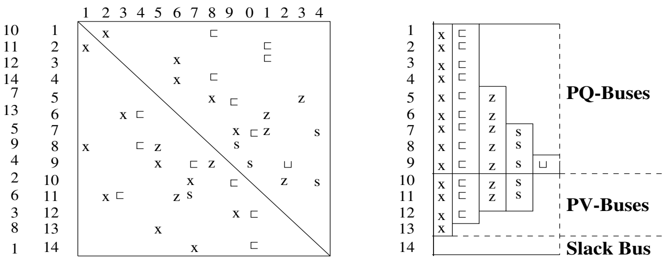

Figure 5: The proposed pseudo-dense–column data storage scheme for the bus admittance matrix of IEEE-14 network.

In the proposed data storage scheme, magnitude and angle *psdcol*'s of the Y_B matrix are stored, consecutively, in vectors YM and δ, respectively, together with their column indices in YCIX. A pointer array PSC contains pointers to the beginning indices of successive *psdcol*'s. It is sufficient to store the row indices of only the first element of each *psdcol* in RPSC. The off-diagonal non-zero elements belonging to the PQ–rows can be accessed as *sub-psdcol*'s by keeping an appropriate pointer vector PQPE which contains pointers to the last non-zero PQ–element of each *psdcol*. These *sub-psdcol*'s, referred here as *PQ-psdcol*'s, will be exploited in the vectorized computation of Eq. 14. For example, in Fig. 5, there are 5 *psdcol*'s of lengths 13, 12, 7, 5, 1 and 5 *PQ-psdcol*'s of lengths 9, 9, 5, 3, 1. The real and imaginary parts of the diagonal elements are stored in arrays G and B permuted according to the proposed ordering. Vectors V and Θ maintain the current bus voltage magnitudes and angles, respectively, according to the new bus ordering. Permutations to B' and B'' bus orderings from the Y_B bus ordering are stored in vectors P1 and P2, respectively. Vectors PS and QS hold the specified real and reactive powers, permuted according to the new ordering. Thus, the last NPV (number of PV buses) entries of the PS vector are constants with values $P_k^s/V_k - G_{kk}V_k$ (see Eq. 13). Arrays α and YMV are scratch arrays used to maintain the arguments of cos / sin factors and $Y_{kh}V_h$ products, respectively. Vectors SP and SQ are also scratch arrays used for real and reactive mismatch computations.

Using this data storage scheme, normalized bus power residual computations can be fully vectorized as follows. In the reactive half-iteration (after solving $B''\Delta V = \Delta Q/V$) voltage magnitudes of PQ buses are updated and normalized power mismatch vectors are initialized as follows:

$$\begin{array}{ll} \text{DO } i = 1, NPQ & \\ \quad \text{V}(i) = \text{V}(i) + \Delta\text{V(P2}(i)) & \\ \quad \text{SP}(i) = \text{PS}(i) \,/\, \text{V}(i) - \text{G}(i) \times \text{V}(i) & \\ \quad \text{SQ}(i) = \text{QS}(i) \,/\, \text{V}(i) + \text{B}(i) \times \text{V}(i) & (15) \\ \text{ENDDO} & \\ \text{DO } i = NPQ{+}1, NBUS & \\ \quad \text{SP}(i) = \text{PS}(i) & (16) \\ \text{ENDDO} & \end{array}$$

DO-loop (15) exploits the vector register re-use capability of vector computers in order to eliminate the re-loading of just stored V values. Then, real off-diagonal contributions are computed and added to the vector SP by executing the following sequence of two DO-loops for each *psdcol* i:

$$\begin{array}{ll} \text{DO } j = \text{PSC}(i), \text{PQPE}(i) & \\ \quad \text{YMV}(j) = \text{YM}(j) \times \text{V(YCIX}(j)) & \\ \quad \text{SP}(j-\Delta) = \text{SP}(j-\Delta) - \text{YMV}(j) \times \cos(\alpha(j)) & (17) \\ \text{ENDDO} & \\ \text{DO } j = \text{PQPE}(i), \text{PSC}(i+1) - 1 & \\ \quad \text{SP}(j-\Delta) = \text{SP}(j-\Delta) - \text{YM}(j) \times \text{V(YCIX}(j)\,) \times & \\ \qquad\qquad \cos(\Theta(j-\Delta) - \Theta(\text{YCIX}(j)) - \delta(j)) & (18) \\ \text{ENDDO} & \end{array}$$

where $\Delta = \text{PSC}(i) - \text{RPSC}(i)$ denotes the constant offset between the indices of the off-diagonal elements in YM, δ arrays and their respective row indices. DO-loops (17) and (18) exploit vector register re-use for the sections of the YMV and YCIX vectors, respectively. The $Y_{kh}V_h$ products corresponding to the entries of PQ-rows are saved in the YMV array for re-use in the real half-iteration. Normalized real power residuals computed in SP are scattered to the appropriate locations of the real power mismatch vector by a single vectorizable scatter DO-loop.

After solving $B'\Delta\Theta = \Delta P/V$, bus voltage angles are updated in Θ vector by a single vectorizable addition DO-loop involving gather operation from $\Delta\Theta$ vector indexing thru permutation vector P1. Then, in the reactive half-iteration, reactive off-diagonal contributions are computed and added, on-the-fly, to SQ by executing the following DO-loop for each *PQ-psdcol* i:

$$\begin{array}{ll} \text{DO } j = \text{PSC}(i), \text{PQPE}(i) & \\ \quad \alpha(j) = \Theta(j-\Delta) - \Theta(\text{YCIX}(j)) - \delta(j) & \\ \quad \text{SQ}(j-\Delta) = \text{SQ}(j-\Delta) - \text{YMV}(j) \times \sin(\alpha(j)) & (19) \\ \text{ENDDO} & \end{array}$$

The angle arguments of the cos / sin factors corresponding to the entries of the PQ-rows are saved for re-use in the real half-iteration. DO-loop (19) exploits vector register re-use for each section of the α vector. Final results accumulated in the SQ vector are scattered to the appropriate locations of the reactive power mismatch vector by a single vectorizable scatter DO-loop.

As seen in DO-loops (15-19), the proposed vectorization scheme minimizes the number of indirect accesses. DO-loops (17-19) verifies that the proposed *psdcol* approach enables the sequential processing of Θ, SP and SQ vectors by avoiding row indexing. During the contribution computations, indirect vector accesses occur only due to the indexing of the Θ and V vectors thru the column index vector YCIX. As seen in DO-loops (15) and (17-19), the proposed vectorization scheme achieves the chaining of contribution computations with the addition of these contributions to SP and SQ vectors. DO-loops (15-16) and (19) show that the proposed *PQ-psdcol* concept avoids the redundant contribution computations for the PV buses without introducing any extra indexing. The same concept is employed to avoid redundant stores into the YMV and α vectors.

In those iterations in which Q–limit enforcement is to be applied, PV buses which violate the reactive power limits are switched to PQ bus type. Hence, a slightly modified version of the proposed vectorization scheme should be executed in those iterations. The sizes of SQ and QS vectors are increased to NBUS$-$1 from NPQ. A new QL vector of size NBUS$-$1 is introduced. The last NPV entries of QS and QL vectors contain the constant $B_{kk}V_k^s$ values and the reactive loads, respectively, for the PV buses. As for the DO-loop modifications, DO-loops (17) and (19) are executed for *all psdcol*'s and DO-loop (18) is removed. The assignment SQ(i) = QS(i) is added to DO-loop (16). The appropriate locations of SP and SQ vectors corresponding to PV buses violating the Q-limits computed incorrectly in the modified DO-loop (16) have to be re-computed in scalar mode. The amount of scalar updates is negligible since the number of such PV buses is much smaller than NBUS in a particular iteration. The modified version of DO-loop (19) accumulates reactive off-diagonal contributions to all buses. Due to the modifications in DO-loops (16) and (19), the last NPV locations of the SQ vector now contain the normalized reactive powers injected to the PV buses. Thus, reactive generations of PV buses are computed in QG by the following vectorizable DO-loop:

$$\begin{array}{ll} \text{DO } i = NPQ + 1, NBUS - 1 & \\ \quad \text{QG}(i) = \text{QL}(i) - \text{SQ}(i) \times \text{V}(i) & (20) \\ \text{ENDDO} & \end{array}$$

Then, reactive power limit check is achieved by performing a single scalar pass over the QG vector. DO-loop (20) shows that the proposed scheme vectorizes the computation of reactive power generations for PV buses without any extra indexing.

5. EXPERIMENTAL RESULTS

In this section, relative performances of the proposed and existing vectorization algorithms on IBM 3090/VF 180S are discussed. These vectorization algorithms are tested using IEEE-118 standard power network and four synthetically generated larger networks with 354, 590, 1180 and 1770 buses. These networks are all obtained by interconnecting the IEEE-118 network.

Table 1 shows the structural properties of the W–partition matrices for B' of the sample networks. We have adopted level-wise partitioning (except the last partition) to benefit from chaining in the FS and BS phases of the proposed vectorization algorithms. ML-MD [15] ordering scheme is used to obtain longer vectors by decreasing the number of levels. Other approaches to increase vector lengths by allowing multi-level partitions with controlled fill-ins after MD-ML, MD-MNP type of orderings make chaining difficult. In Table 1, M_f denotes the percent increase in level-wise partitioned W fill-ins introduced by ML-MD ordering with multi-level last partition instead of MD ordering. Table 1 shows that the adopted partitioning scheme introduces roughly 10% fill-in increase for the sake of efficient vectorization. In the same table, n_ℓ and n_p denote the number of levels and partitions, respectively.

The total amount of start-up time overhead is proportional to the number of sections processed. Table 1 confirms the expectation that SVPP schemes process considerably smaller number of sections than MVPP schemes. Note that, the same number of sections is processed in both FS and BS phases in SVPP schemes. By construction, the lengths of *psdcol*'s in a particular W^t partition is limited to the number of buses in that partition. However, W partition matrices may have much longer *psdcol*'s. That is why the number of sections in the BS phase of MVPP methods is considerably greater than that of the FS phase as illustrated in Table 1. The increase in the number of sections processed in the FS phase of PR4 scheme compared to PR2-3 schemes results from the doubling of the number of DO-loop executions as explained earlier. Experimental results show that vectorizable DO-loops of length shorter than some critical number yield better performance if executed in scalar mode rather than vector mode. Current implementation detects last sections of length shorter than 20 and enforces them to scalar execution. In this work, level-wise vector lengths are checked against this critical number (20), starting from the first level towards the last one until a vector of smaller length is encountered. Then, the current level and the rest are included in the last partition.

Table 2 illustrates the number of redundant scalar additions introduced in order to vectorize the addition operations in SVPP methods. Comparison of GR1 and PR1 columns reveals that the proposed round-robin re-ordering algorithm exploiting intra-section recurrence concept reduces the number of scalar additions drastically. The proposed re-ordering algorithm is expected to yield much better performance for smaller section sizes, e.g., K=64, as is shown in parenthesis in this table. The number of scalar additions in the FS phase is much smaller than that of the BS phase due to greater number of recurrent column indices than recurrent row indices in partition matrices. Among all MVPP methods, the only redundancy occurs in the FS phase of PR3. Recall that PR3 introduces this redundancy in order to reduce the number of indirect element selections. Table 2 shows that the redundancy in PR3 which is equal to twice the sum of the length of first *psdcol* in each partition, is larger than M. However, this corresponds to a delivery cycle overhead of only $\approx$ 13% compared to PR2.

Table 1: The number of off-diagonal non-zero elements, levels, partitions, and sections for B' matrices for sample networks.

					No. of Sections			
					SV PP	MVPP		
						BS	FS	
NBUS	M	M_f	n_ℓ	n_p			PR2-3	PR4
354	922	11.1	17	8	10	40	24	39
590	1557	9.5	17	10	16	60	31	47
1180	3270	11.4	25	15	34	146	51	68
1770	4877	10.0	27	17	45	159	69	97

Table 2: The number of redundant operations in the FS and BS phases of different schemes.

		SVPP				MVPP
		scalar additions				vectorized load/stores
		BS Phase		FS Phase		FS Phase
NBUS	M	GR1	PR1	GR1	PR1	PR3
118	299	115	115	95	95(48)	232
354	922	501	250	360	134(63)	948
590	1557	889	342	603	149(51)	1708
1180	3270	2020	560	1372	198(89)	3598
1770	4877	3039	688	2030	199(83)	5528

Tables 3 and 4 illustrate the performances of various vectorization schemes for the FBS phase. The last column of Table 3 shows the execution time of DS phase for all schemes. As seen in Tables 3 and 4, PR1 outperforms GR1 due to both the chaining and the substantial reduction in the number of redundant scalar additions achieved by the proposed re-ordering algorithm. Table 3 confirms the expectation that the BS phase of PR3-4 schemes is the best among all methods due to the minimized number of indirections. Table 4 illustrates that the proposed SVPP method PR1 is the best method among all schemes in the FS phase due to the considerably smaller vector lengths in MVPP methods for the small-to-medium size networks (118, 354, 590). The proposed MVPP scheme PR2 performs better in the FS phase for the larger networks (1180, 1770) due to increased vector lengths. In the FS phase, the proposed PR3 scheme does not perform as expected because of the redundancy mentioned earlier. The relative performance of PR4 over PR2 in the FS phase is expected to increase with increasing problem size due to its smaller number of indirections and larger number of sections. Unfortunately, on IBM 3090, PR4 causes more cache misses than PR2 for large size networks due to increased number of vectors used in the data storage scheme. Hence, PR4 scheme must be experimented against PR2 on vector computers which do not utilize cache hierarchy such as Cray.

Table 3: Execution times in microseconds for the BS and DS phases of different schemes for the solution of $B'\Delta\Theta = \Delta P/V$.

Network Size	Execution times in microseconds					
	SVPP		MVPP			DS
NBUS	GR1	PR1	GR2	PR2	PR3-4	
118	143	122	149	133	77	15
354	385	300	401	348	174	39
590	566	455	579	499	283	62
1180	1038	837	1071	900	600	121
1770	1578	1245	1327	1130	789	163

Table 4: Execution times in microseconds for the FS phase of different schemes for the solution of $B'\Delta\Theta = \Delta P/V$.

Network Size	Execution times in microseconds					
	SVPP		MVPP			
NBUS	GR1	PR1	GR2	PR2	PR3	PR4
118	138	120	155	137	139	146
354	361	282	359	311	312	329
590	501	424	519	443	462	470
1180	887	781	864	742	845	833
1770	1340	1161	1040	913	1172	1074

Table 5 provides the execution times for one iteration of the mismatch computation phase using two different approaches. The symmetric version (sym), which exploits the symmetry of the Y_B matrix, corresponds to the implementation of the scheme proposed by Granelli et al. [11]. The one which does not exploit the symmetry (no sym) corresponds to the implementation of the proposed scheme explained in Section 4. Table 5 confirms the expectation that the symmetric approach performs better in scalar mode due to the considerably smaller number of expensive cos / sin computations. However, the scalar performance difference between these two approaches is substantially small due to larger number of indirections in the symmetric method. The proposed vectorization scheme performs better than the symmetric one except for the smallest network. The proposed scheme achieves this good performance by minimizing the number of indirections which is very important in efficient vectorization. As seen in Table 5, the proposed vectorization scheme outperforms the symmetric approach, in the absence of Q-limit check, by avoiding redundant computations for PV buses without introducing any extra indexing overhead. The proposed vectorization scheme still performs better than the symmetric one in the presence of Q-limit check. Table 5 confirms the general fact that best scalar algorithm may not lead to the best vectorization algorithm.

The timing results illustrated for the vector performance of FBS phase in Table 6 are calculated from the best attained results of FS and BS phases for each network. The scalar FBS timing results correspond to the scalar execution of PR1 without redundant additions where MD scheme is adopted. The scalar and vector mismatch computation timing results correspond to the best scalar and vector executions in the absence of Q-limit check, respectively. Table 6 illustrates that the speed-up increases with increasing problem size.

Table 5: Execution times in microseconds for the mismatch computation phase for different schemes.

	Scalar		Vector			
					Q-limit check	
NBUS	sym	no sym	sym	no sym	sym	no sym
118	1340	1392	786	900	814	1077
354	4070	4153	2232	1971	2308	2447
590	6708	6962	3631	3086	3857	3833
1180	13654	14065	7420	5866	7955	7525
1770	20914	21343	11194	8188	11957	10502

Table 6: Execution times in microseconds of the best scalar and vectorized schemes and the speed-up for the FBS and mismatch computation phases.

	FBS ($B'\Delta\Theta = \Delta P/V$)			Mismatch Computation		
NBUS	Sca	Vec	Speedup	Sca	Vec	Speedup
118	268	212	1.26	1340	786	1.70
354	845	495	1.71	4070	1971	2.06
590	1392	769	1.81	6708	3086	2.17
1180	2990	1462	2.05	13654	5866	2.33
1770	4469	1865	2.40	20914	8188	2.55

6. CONCLUSION

This paper presents novel data storage schemes and algorithms for the efficient vectorization of repeated sparse power system network computations. The proposed algorithms resolve the recurrence problem, exploit chaining and sectioning, and minimize the number of indirect element selections to attain utmost vector performance. The solution phase of FDLF, which involves the repeated solution of linear system of equations and power mismatch computations, is used for benchmarking the proposed vectorization schemes. The relative performances of the proposed and existing vectorization schemes are evaluated, both theoretically and experimentally on IBM 3090/VF. Results demonstrate that the proposed schemes perform better than the existing vectorization schemes.

REFERENCES

References

[1] IEEE Committee Report, "Parallel Processing in Power Systems Computation," *IEEE Trans. on Power Systems*, Vol. 7, No. 2, pp. 629–638, May 1992.

[2] Betancourt, R., and Alvarado, F.L., "Parallel Inversion of Sparse Matrices," *IEEE Trans. on Power Systems*, Vol. 1, No. 1, pp. 74–81, February 1986.

[3] Abur, A., "A Parallel Scheme for the Forward/Backward Substitutions in Solving Sparse Linear Equations," *IEEE Trans. on Power Systems*, Vol. 3, No. 4, pp. 1471–1478, November 1988.

[4] Enns, M. K., Tinney, W. F., and Alvarado, F. L., "Sparse Matrix Inverse Factors," *IEEE Trans. on Power Systems*, Vol. 5, No. 2, pp. 466–472, May 1990.

[5] Gomez, A., and Betancourt, R., "Implementation of the Fast Decoupled Load Flow on a Vector Computer," *IEEE Trans. on Power Systems*, pp. 977–983, Feb. 1990.

[6] Alvarado, F.L., Yu, D.C., and Betancourt, R., "Partitioned Sparse A^{-1} Methods," *IEEE Trans. on Power Systems*, Vol. 5, No. 2, pp. 452–459, May 1990.

[7] Granelli, G. P., Montagna, M., and Pasini, G.L., "Efficient Factorization and Solution Algorithms on Vector Computers ," *Electric Power Systems Research*, Vol. 20, No. 2, pp. 121–136, 1991.

[8] Granelli, G. P., Montagna, M., Pasini, G. L., Marannino, P., "Vector Computer Implementation of Power Flow Outage Studies," *IEEE Trans. on Power Systems*, Vol. 7, No. 2, pp. 798–804, May 1992.

[9] Lau, K., Tylavsky, D. J., and Bose, A., "Coarse Grain Scheduling in Parallel Triangular Factorization and Solution of Power System Matrices," *IEEE Trans. on Power Systems*, Vol. 6, No. 2, pp. 708–714, May 1991.

[10] Padilha, A., and Morelato, A., "A W-Matrix Methodology for Solving Sparse Network Equations on Multiprocessor Computers," *IEEE Trans. on Power Systems*, Vol. 7, No. 3, pp. 1023–1030, August 1992.

[11] Granelli, G. P., Montagna, M., Pasini, G. L., and Marannino, P., "A W-Matrix Based Fast Decoupled Load Flow for Contingency Studies on Vector Computer," *IEEE Trans. on Power Systems*, Vol. 8, No. 3, pp. 946–953, August 1993.

[12] Stott, B., and Alsac, O., "Fast Decoupled Load Flow," *IEEE Trans. on Power App. Syst.*, Vol. 73, pp. 859–867, May/June 1974.

[13] Anderson, D. M., and Wollenberg, B. F., "Power System Steady State Security Analysis Using Vector Processing Computers," *IEEE Trans. on Power Systems*, Vol. 7, No. 4, pp. 1451–1455, November 1992.

[14] Chan, S. M., and Brandwajn, V., "Partial Matrix Refactorization," *IEEE Trans. on Power Systems*, Vol. 1, No. 1, pp. 193–200, February 1986.

[15] Betancourt, R., "An Efficient Heuristic Ordering Algorithm for Partial Matrix Refactorization," *IEEE Trans. on Power Systems*, Vol. 3, No. 3, pp. 1181–1187, August 1988.

[16] Gomez, A., and Franquela, L. G., "An Efficient Ordering Algorithm to Improve Sparse Vector Methods," *IEEE Trans. on Power Systems*, Vol. 3, No. 4, pp. 1538–1544, November 1988.

Cevdet Aykanat received the B.S. and M.S. degrees from the Middle East Technical University, Ankara, Turkey, and Ph.D. degree from The Ohio State University, Columbus, all in EE. He was a Fulbright scholar during his Ph.D, studies. He worked at Intel Supercomputer Systems Division, Oregon. Since 1988 he has been with the Computer Engineering Dept., Bilkent University, Ankara, Turkey. His research interests include parallel computer architectures and algorithms, neural algorithms.

Özlem Özgü received the BS(1987) and MS(1990) degrees from the Middle East Technical University, Ankara, Turkey, all in Computer Engineering. Since 1990, she has been a research assistant with the Computer Engineering Dept., Bilkent University, Ankara, Turkey. Her research interests include parallel computer architectures and algorithms.

Nezih Güven(M'86) received the BSEE(1979) degree from Middle East Technical University, Ankara, Turkey and MS(1981) and Ph.D.(1984) degrees in EE from The Ohio State University, Columbus. From 1984 to 1985, he was with the Dept. of Electrical Engineering and Systems Science at Michigan State University. Since 1986, he has been at Middle East Technical University. His research interests include computer applications in power systems and distribution automation.