EFFICIENT VECTORIZATION OF FORWARD/BACKWARD SUBSTITUTIONS IN SOLVING SPARSE LINEAR EQUATIONS

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

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

Abstract- Vector processors have promised an enormous increase in computing speed for computationally intensive and time-critical power system problems which require the repeated solution of sparse linear equations. Due to short vectors processed in these applications, standard sparsity-based algorithms need to be restructured for efficient vectorization. This paper presents a novel data storage scheme and an efficient vectorization algorithm that exploits the intrinsic architectural features of vector computers such as sectioning and chaining. As the benchmark, the solution phase of the Fast Decoupled Load Flow algorithm is used in simulations. The relative performances of the proposed and existing vectorization schemes are evaluated, both theoretically and experimentally, on IBM 3090/VF.

1. INTRODUCTION

Most power system problems such as load flow, state estimation, transient stability, etc. require repetitive solution of a set of sparse linear equations of the form

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

The standard procedure for solving these equations consists of two main phases: factorization of the coefficient matric A into LDL^T form, and forward/backward substitutions (FBS). Although the heavy computational effort associated with such repetitive solutions has been greatly reduced by using sparse matrix techniques, it is still very time consuming. Recent developments in computer technology suggest that further reductions in computation time may be achieved through vector processing. However, standard sparsity-based algorithms used in power system applications need to be restructured for efficient vectorization due to extremely short vectors processed.

Vector processing achieves improvement in system through-put by exploiting pipelining. To achieve pipelining, an operation is divided into a sequence of substasks, 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 over-head 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 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. For example, K=128 in IBM 3090/VF. 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 sectioning 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 results 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
  B(j)=B(j)+V(IX(j))xW(j)                    (2)
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 loadded, stored or processed directly in an arbitrary sequence. In indirect addressing, the memory location of the vector elements to be accessed is indicated by a vector of integer indices, which must be previously stored in a vector register. In DO-loop (2), vectors W, B 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
  B(IX(j))=B(IX(j))+W(j)                     (3)
ENDDO
```

This DO-loop contains apparent dependence due to indexing of the B array by the IX array in both sides of the statement in (3). 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 the FBS of sparse linear equations, the recurrence problem is a crucial bottleneck for efficient vectorization. The DO-loop (3) 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.

This paper presents a novel data storage scheme and a vectorization algorithm that resolves the recurrence problem and exploits the intrinsic architectural features of vector computers such as chaining and sectioning. The solution phase of the Fast Decoupled Load Flow (FDLF) algorithm [1], which is the most popular method frequently utilized by power utilities, is used as the benchmark for the proposed algorithm. In the solution phase of FDLF analysis, real and reactive load flow equations; $B'\Delta\theta=\Delta P/V$ and $B''\Delta V=\Delta Q/V$ are repeatedly solved where B' and B'' are randomly sparse Jacobian matrices.

2. W-MATRIX APPROACH

The FBS phase in the solution of linear system of equations consists of the following steps:

$$\text{(a) } Lz=b; \qquad \text{(b) } Dy=z; \qquad \text{(c) } L^Tx=y \tag{4}$$

where L and D are factor matrices of the coefficient matrix. Diagonal Scaling (DS) step (4.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 (4.a) and Backward Substitution (BS) step (4.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 Ax=b can be computed as

$$\text{(a) } z=Wb; \qquad \text{(b) } y=D^{-1}z \qquad \text{(c) } x=W^Ty \tag{5}$$

0-7803-1772-6/94/$3.00

Here, $W=L^{-1}$ is called the inverse-factor. The advantage of (5) over (4) 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 [2].

In partitioning schemes, the factor L is expressed as $L=L_1....L_N$, where the elemental factor matrix L_i is an identity matrix expect for the i-th column which contains the corresponding column of L. Thus, $W=W_N...W_1$ 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_{p1}, L_{p2},...L_{pk} so that $W=W_{pk}...W_{p2}W_{p1}$. Hence Eq. (5) is transformed into

$$(a)z=W_{pk}...W_{p1}b; \quad (b)y=D^{-1}z; \quad (c)x=W_{p1}{}^{T}...W_{pk}{}^{T}y \qquad (6)$$

Various algorithms have been proposed for determining inverse factor matrix partitioning which produce zero or only a pre-fixed maximum number of fill-ins [2, 3]. 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, etc., have also been proposed to reduce the total number of levels in the resulting FPG.

3. VECTORIZATION OF FBS

In this section, the implementation of the vectorization approaches proposed by Gomez et al [4] and Granelli et al [5] will be briefly discussed. These schemes store the non-zero elements of W-partition matrices column-wise, in partition order, in WV together with their row and column indices in RIX and CIX, respectively. Figure 2 illustrates the data storage schemes utilized 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 the W-partition matrices as well.

The FS phase of the approach proposed by Gomez [4] involves the following two DO- loops for each partition i:

```
DO  j=PP (i), PP(i+1)-1
  WVR(j)=WV(j)xBV(CIX(j))                    (7.a)
ENDDO
DO  j=PP(i), PP(i+1)-1
  BV(RIX(j)=BV(RIX(j))+WVR(j)                (7.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 6.a) on which the solution (z in 6.a) is rewritten. DO-loops (7.a) and (7.b) perform the multiplication and addition operations involved in each sparse matrix-vector product in (6.a). respectively. The DO-loop structure of the BS phase can easily be obtained by interchanging CIX with RIX in (7). In the FS (BS) phase, the addition DO-loop (7.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 DO-loop (7.a) is vectorized on IBM 3090/VF by the following sequence of vector instructions: VL (Vector Load), VLID (Vector Load Indirect), VM (Vector Multiply) and VST (Vector Store). The overall computational complexity of the vectorized multiplication operations involved in FS and BS phases is $10M + 8St_s$ where S is the total number of partition-basis sections in W array and t_s is the start-up time overhead. The addition DO-loop (7.b) is executed in scalar mode and 2M operations are involved. This scheme requires two real (WV, WVR) and two integer (CIX, RIX) vectors, of lengths M, and one integer vector of PP of length n_l-1. Hence, the storage complexity of this scheme is $4M+4n_l \approx 4M$

The scheme proposed by Granelli et al. [5] 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). Using this storage scheme, the implementation of Granelli's method for the FS phase can be obtained by replacing the addition DO-loop (7.b) by the following two DO-loops.

```
DO j=PP(i), PP(i+1)-1
  BV(RIXRF(j))=BV(RIXRF(j))+WVR(j)           (8.a)
ENDDO
DO r=RRPP(i), RRPP(i+1)-1
  BV(RRIX(r))=BV(RRIX(r))+WVR(RRIXIX(r))     (8.b)
ENDDO
```

The DO-loop structure of the BS phase is similar. This scheme will be referred to as GR 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 (8.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 (8.b).

The vectorized addition DO-loop (8.a) in FS and BS phases is implemented by the following sequence of vector instructions; VL (Vector Load), VLID (Vector Load InDirect), VADD (Vector ADD), VSTID (Vector Store InDirect). Considering the sectioning of vector operations, this DO-loop requires $4S_l$ vector instructions and $6m_l+ 4S_lt_s$ machine cycles in the l-th iteration. Here $S_l=\lfloor m_l/k \rfloor$ and m_l is the number of nonzero elements in the l-th partition. Thus, the overall complexity of the vectorized solution is $22M + 16St_s$ machine cycles.

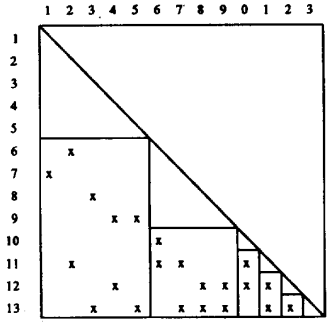

Figure 1: The sparsity structure of the factor and W-partition matrices of the B' matrix.

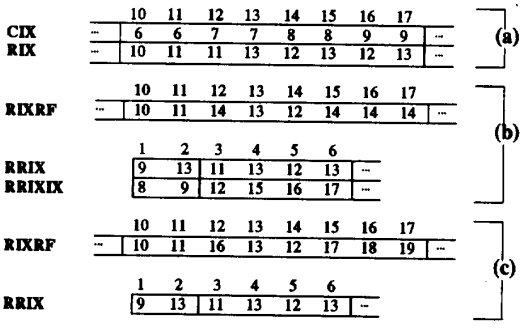

Figure 2: The data storage schemes for the FS phase: (a) GB, (b) GR, (c) PR

Proposed Algorithm

Although scheme GR 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. Here, we propose an efficient scheme to resolve the recurrence problem which also enables chaining. In scheme GR, all multiplication results are saved in a temporary array WVR so that multiplication results corresponding to 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 PR, 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 array 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 GR 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)xBV(CIX(j))          (9.a)
ENDDO
DO r=RRPP(i), RRPP(i+1)-1
  BV(RRIX(r))=BV)RRIX(r))+BV(N+r)                     (9.b)
ENDDO
```

The DO-loop structure of the BS phase is similar. The DO-loop (9.a) achieves the chaining of addition and multiplication operations. Due to chaining , 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 (9.a) contains two types of apparent dependencies. The first is through indexing of the BV array by the RIXRF vector in both sides of (9.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 (9.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 vectorize DO-loop (9.a) to achieve chaining.

The compund DO-loop (9.a) is implemented on IBM 3090/VF by the following sequence of vector instructions: VLID (Vector Load InDirect), VL (Vector Load), VLID (Vector Load Indirect), VMAD (Vector Multiply and ADd), VSTID (Vector STore InDirect). In the l-th iteration, this vectorized loop requires $9m_l + 6S_l t_s$ machine cycles. The overall computation complexity of the FS and BS phases is $18M + 12St_s$ machine cycles. The proposed scheme eliminates the need for using the temporary real array WVR through chaining. However, the length of the BV array is increased by R.

Table 1 illustrates the storage and computational complexity of the proposed scheme PR compared with GB and GR schemes. Storage complexities are given in terms of words and they denote the asymptotic complexities. Computational vector complexities denote the total number of machine cycles requred to execute the vectorized operations. Computational scalar complexities are given in terms of the total number of scalar additions required by each scheme.

Table 1. Storage and Computational Complexity of Different Vectorization Schemes on IBM 3090/VF

Scheme	Storage (words)	Computational	
		Vector (m/c cycles)	Scalar (no. of adds)
GB	4M	$10M + 8St_s$	2M
GR	6M + 2R	$22M + 16St_s$	R
PR	5M + 2R	$18M + 12St_s$	R

Note that, the computational vector complexities of both Granelli's and the proposed scheme are approximately twice that of Gomez's scheme. Hence both Granelli's and the proposed schemes are expected to yield better performances than Gomez's scheme if R<2M.

The proposed scheme PR achieves substantial performance improvement in vectorization over scheme GR through chaining. For example, on IBM 3090/VF, PR 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 (9.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 (8.b) of GR 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 GR scheme in processing the recurrent elements.

The Last Partition

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 (9.a), and hence, substantially reduces the total number of redundant 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 GR and PR 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 [6]. 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.

Intra-/Inter-Section Recurrences

Consider a multi-section level with m non-zero elements, so that the number of sections, $s=\lceil m/k \rceil > 1$. 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 can be classified as inter-section which 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 apperances 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 PR scheme to $\lfloor d_{ix}/s \rfloor$-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.

Figure 3 shows the round-robin allocation of a column c with d_c=5 non-zero elements in a partition with m=249 non-zero elements and S=[249/100]=3 sections. The section size is assumed to be K=100, and partition-basis local indices are used. In Figure 3(a), the last non-zero element of the previous column c-1 is assumed to be assigned to the first section with local index 40. Shaded portions in Fig. 3(a) denote the locations already allocated by the round-robin algorithm for the non-zero elements of the previous columns in that partition. As seen in Fig.3(a), the proposed scheme assigns recurrent elements to different sections, in a round-robin fashion, to minimize the number of intra-section recurrences in the RIX (CIX) array. All first recurrences associated with the same index in different sections can be assigned the same extended B location B(N+r). Similarly, all second, third, ..., etc. occurances in *different* sections can be assigned with the same extended B locations. The proposed re-ordering algorithm easily detects multi-intra-section recurrences aswell. Figure 3(b) illustrates this concept for the allocation instance given in Fig. 3(a). The number of scalar additions required for the recurrent column index c is reduced from 2 *in Fig. 3(a) to 1 in Fig. 3(b).*

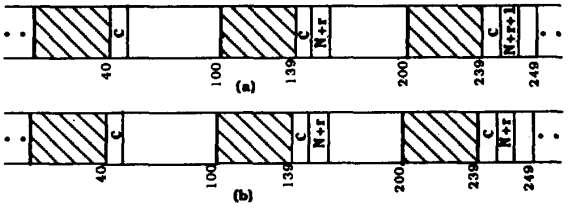

Figure 3. The proposed round-robin allocation scheme

4. EXPERIMENTAL RESULTS

In this section, relative performances of the proposed and existing vectorization algorithms are tested for the solution phase of IEEE-118 standard power network and four synthetically generated larger networks with 354, 590, 1180 and 1770 buses.

Table 2 shows the structural properties of the inverse-factor 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 ordering scheme is used to obtain longer vectors by decreasing the number of levels. In Table 2, M_f denotes the percent increase in level-wise partitioned W fillins introduced by ML-MD ordering with multi-level last partition instead of MD ordering. Table 2 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. Note that, the same number of sections is processed in both FS and BS phasese. 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 shorter than 20 and enforce 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: The number of off-diagonal non-zero elements, levels, partitions, and sections for B' matrices of sample networks.

NBUS	M	M_f	n_l	n_p	No of sections
354	922	11.1	17	8	10
590	1557	9.5	17	10	16
1180	3270	11.4	25	15	34
1770	4877	10.0	27	17	45

Table 3 illustrates the number of redundant scalar additions introduced in order to vectorize the addition operations. Comparison of GR and PR columns reveals that the proposed round-robin re-ordering algorithm exploiting multi-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 paranthesis 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.

Table 4 illustrates the performances of GR and PR schemes for the FBS phase. The fourth column of Table 4 shows the execution time of DS phase for all schemes. As seen in Table 4, PR outperforms GR due to the substantial reduction in the number of redundant scalar additions achieved by the proposed re-ordering algorithm. The speed-up obtained with PR against scalar execution is between 1.25 and 2.0 and it increases with increasing problem size.

Table 3. The Number of Redundant Operations in the FS and BS phases of different schemes

		scalar additions			
		BS Phase		FS Phase	
NBUS	M	GR	PR	GR	PR
118	299	115	115	95	95(48)
354	922	501	250	360	134(63)
590	1557	889	342	603	149(51)
1180	3270	2020	560	1372	198(89)
1770	4877	3039	688	2030	199(83)

Table 4. Execution times for the BS, DS and FS phases of $B'\Delta\theta = \Delta P/V$

Network Size	Execution times in microseconds				
NBUS	BS Phase		DS	FS Phase	
	GR	PR		GR	PR
354	385	300	39	361	282
590	566	455	62	501	424
1180	1038	837	121	887	781
1770	1578	1245	163	1340	1161

5. CONCLUSION

This paper presents a novel data storage scheme and algorithm for the efficient vectorization of the forward/backward substitutions in the solution of linear system of equations arising in Fast Decoupled Load Flow. The proposed algorithm resolves the recurrence problem and exploits chaining and sectioning. 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 much better than existing vectorization schemes.

References

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

[2]. 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

[3]. Alvarado, F.L., Yu, D.C., and Betancourt, R., "Partioned Sparse A^{-1} Methods", Vol. 5, No.2, pp. 452-459, May 1990.

[4]. 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.

[5]. 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.

[6]. Padilha, A., 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.