Society for Science and Education United Kingdom
TNC Transactions on Networks and Communications
Volume 2, Issue 4
ISSN: 2054 - 7420

Gain Matrix Distributed Computing Technique for Power System State Estimation

[1]H. Nagaraja Udupa, [2]H. Ravishankar Kamath
[1]Mewar University, District Mewar, Chittograh, Rajasthan, India
[2]Malwa Institute of Technology, RGPV University, Indore, India
[1] hnudupa@gmail.com; [2] rskamath272@gmail.com

ABSTRACT

The Electric Power System State Estimation problem involves large sparse matrices. The Jacobian matrix is highly sparse in nature and the computational efforts can be enhanced by avoiding arithmetic operations resulting in 'zero'. The researchers have introduced sparse matrix techniques so as to store only non-zero elements of the matrix and thereby reducing the huge dynamic memory requirements, which intern reduce the computational time. A few such techniques [2],[3], [4] are listed in the reference. The primary focuses of these sparse techniques are on the memory/storage space reduction.

This paper elaborates a different technique to obtain the "effective operation" with the focus on the computational time and the storage space reduction. The "effective operation" can be achieved without applying conventional compact storage techniques to find the Jacobian product. A different style for multiplication of two large sparse Jacobian matrices is adopted to obtain this novel approach. As a result, computational time is reduced and also Jacobian array size is reduced form two dimensional array to single dimensional array. The solution gives scope for distributed/parallel computing without disturbing the network structure [6].

Key Words: SE - State Estimation, WLS – Weighted Least Square, NR – Newton Rapson, ISE – Integrated State estimation, 'A' gain matrix, , NA – Node Area- A node along with its connected Node is referred as Node Area, H1 to H12 are the sub set of Jacobian metrics 'J'.

1 Introduction

The state of power system is to be known for healthy planning, operation and energy management for both online and off-line system. The complex non-linear equations along with large sparse matrix involved in the power system makes it complicated and difficult for fast computation of state variables. Many researchers have presented different technique to overcome this problem. A few such papers [5], [7] & [8] are given in reference. Interestingly, there is a one to one relation between the network incident matrix and the Jacobian matrix for

DOI: 10.14738/tnc.24.311
Publication Date: 4th August 2014
URL: http://dx.doi.org/10.14738/tnc.24.311

non-zero elements. Using this intelligence it is possible to focus all the computations only for non-zero elements and thereby minimize the computational time with minimum dynamic storage space. It is essential to modify the NR solution steps to achieve the same. An insight of the procedural steps involved in the existing NR method is given below for better understanding of the modification detailed out in the later session.

1.1 State Estimation:- Newton-Raphson technique

By applying the tylor series to the nonlinear equations of power system following equations is derived [1].

$(J^T W J)\ \Delta x = J^T W\ \Delta z$

$A = (J^T WJ)$ & $b = J^T W\ \Delta z$

$A\ \Delta x = b$ (1)

x_i– No of state variables :- = (2*n-1)

n – No.of network nodes : - = 1,2,…n.

m– Total no of measurements

J – Jacobian matrix, size is : - m*(2n-1)

W - Diagonal weigh matrix of the order of (m*m)

A – Gain matrix of the order of (2n-1)*(2n-1)

$[x]^T = [\delta_1, \delta_2, ...\delta_{n-1}; v_1, v_2, ...v_n,]$; of the order of *x* is (2n-1)*1.

$[z^{measured}]^T = [P_i, Q_i, p_{ij}, q_{ij}, V_i, \delta_i]$; of the order of (m*1); These measurements may include one or all quantities.

Pi ,Qi = Real & Imaginary part of injected power respectively.

pij ,qij = Real & imaginary part of line follows respectively

$\Delta z = z^{measured} - z^{calculated}$, size is: - m*1;

$b = J_0^T W\ \Delta z$ of the order of (2n-1)*1;

$$\overbrace{\begin{bmatrix}\Delta P_i\\ \Delta Q_i\\ \Delta p_{ij}\\ \Delta q_{ij}\\ \Delta v_i\\ \Delta\partial_i\end{bmatrix}}^{[\Delta z_i]} \Longrightarrow \overbrace{\begin{bmatrix} |H_1| & |H_2| \\ |H_3| & |H_4| \\ |H_5| & |H_6| \\ |H_7| & |H_8| \\ |\ 0\ | & |\ 1\ | \\ |\ 1\ | & |\ 0\ | \end{bmatrix}}^{[J]} \overbrace{\begin{bmatrix}\Delta\partial_i\\ \Delta v_i\end{bmatrix}}^{\Delta x_i} \qquad (2)$$

This is a set of linear equations, if higher order terms of the taylor expansion of f(x) were really negligible, the solution yield the correct ‘x’. The state variable vector x is obtained by solving the equation A*Δx = b iteratively. The vector x should therefore be changed accordingly after every iteration till the convergence is obtained.

$x^{c+1} = x^c + \Delta x^c$; ‘c’-iteration count. Elements of Jacobean are derived from injected power, and line flow equations.

1.2 Conventional Computing Steps

```
Step 1: - Read input data
Step 2: - From system parameters find $Y_{bus}$
Step 3: - Initialize $[v_i]^T = 1$; and $[\partial_i]^T=0$;
Step 4: - Find $[z]^{cal}$ and $[\Delta z]= [z]^{measured} -[z]^{cal}$
Step 5: - Find all rows of [J] and $[\Delta z]$
Step 6: - Find $[J]^T*[W]*[J] =[A]$ and $[J]^T*[W]*[\Delta z] = [b]$
Step 7: - Find $[\Delta x_i] = [A]^{-1}*[b]$;
Step 8: - check for $[\Delta x_i]$
    if $[\Delta x_i]<<€$;
    if No - Update [xi]; $[xi]^{new} = [x_i]^{old} +[\Delta x_i]^{old}$; then repeat from step 4.
    if yes – Stop
```

2 Distributed Technique: - New Method

The dimension/size of the matrices involved in equation (1) depends on the number of network nodes. The conventional algorithm can be divided into two parts,

i. up to the formation of matrix 'A' and 'b'
ii. obtaining the solution for $[\Delta xi] = [A]^{-1}*[b]$

It is difficult to allot multiple processors to solve the problem in its present form. A new novel distributed computing technique is discussed to address the issue.

2.1 Matrix Multiplication – Alternate Technique

Let us consider the conventional way of multiplication of two matrices.

$$[C]*[D] = [A]; \quad a_{\substack{ij \\ i=1,2,\dots.n \\ j=1,2,\dots r}} = \sum_{k=1}^{m} (c_{ik} * d_{kj}) \tag{3}$$

Order of 'C' is : – (n*m), Order of 'D' is : – (m*r) and Order of 'A' is – (n*r)
From the (3), the resultant matrix can be rearranged as shown below

$$A = \sum_{k=1}^{m} \left\{ \overset{k=1}{\begin{pmatrix} (c_{i1} * d_{1j}) \\ i = 1,2,\dots.n; \\ j = 1,2,\dots r \end{pmatrix}} \overset{k=p}{\begin{pmatrix} \\ \\ \end{pmatrix}} \overset{k=m}{\begin{pmatrix} (c_{im} * d_{mj}) \\ i = 1,2,\dots.n; \\ j = 1,2,\dots r \end{pmatrix}} \right\} \tag{4}$$

Each sub-matrices of 'A' can be obtained by taking a Coolum of 'C' and corresponding row of

'D', for example Am is obtained by multiplying the mth column of 'C' and mth row of 'D', which is as follows

$$[A_m] = \begin{bmatrix} c_{1m} \\ c_{2m} \\ . \\ . \\ c_{nm} \end{bmatrix} \begin{bmatrix} d_{m1} & d_{m2} & \ldots d_{mr} \end{bmatrix}; \tag{5}$$

.

$$[A] = [A_1]_{k=1} + [A_2]_{k=2} + \ldots\ldots + [A_m]_{k=m} = \sum_{k=1}^{m} A_m \tag{6}$$

2.2 Distributed Approach

From the above relationship, the components of 'A' can be computed by considering a Column of 'C' and corresponding row of 'D'. By applying the above relationship to $[J]^T*[W]*[J] = [A]$ and $[J]^T*[W]*[\Delta z] = [b]$, yields, ('W' assumed unity diagonal)

$$A = \sum_{k=1}^{m} \left\{ \overset{k=1}{\begin{pmatrix} (J_{i1}^T * J_{1j}) \\ i = 1,2,\ldots.2n-1; \\ j = 1,2,\ldots 2n-1 \end{pmatrix}} \cdots \overset{k=m}{\begin{pmatrix} (J_{im}^T * J_{mj}) \\ i = 1,2,\ldots.2n-1; \\ j = 1,2,\ldots 2n-1; \end{pmatrix}} \right\}$$

$$b = \sum_{k=1}^{m} \left\{ \overset{k=1}{\begin{pmatrix} (J_{i1}^T * \Delta z_1) \\ i = 1,2..(2n\text{-}1) \end{pmatrix}} \cdots \overset{k=m}{\begin{pmatrix} (J_{im}^T * \Delta z_m) \\ i = 1,2..(2n\text{-}1) \end{pmatrix}} \right\}$$

$$[A_m] = \begin{bmatrix} J_{m1} \\ J_{m2} \\ . \\ . \\ J_{m(2n-1)} \end{bmatrix} \begin{bmatrix} J_{m1} & J_{m2} & \ldots J_{m(2n-1)} \end{bmatrix}$$

$$[\mathrm{b_m}] = \begin{bmatrix} J_{m1} \\ J_{m2} \\ . \\ . \\ J_{m(2n-1)} \end{bmatrix} [\Delta \mathrm{z_m}]$$

$$[A] = [A_1]_{k=1} + [A_2]_{k=2} + \dots\dots + [A_m]_{k=m} \tag{6}$$

$$\text{and } [b] = [b_1]_{k=1} + [b_2]_{k=2} + \dots\dots + [b_m]_{k=m} \tag{7}$$

It is evident from the above relationship that after obtaining the each row of Jacobin with respect to a measurement, corresponding sub-matrices of resultant matrix (A and b) can be obtained. Hence it is not necessary to form the complete Jacobian matrix before multiplication of $[J]^T$ *W*[J] =[A] and $[J]^T$*W*[Δz] = [b]. It reduces the Dynamic size of 'J' matrix from m*(2n-1) to 1*(2n-1). Also A_1, A_2.. A_m and b_1, b_2… b_m can be formed independently. The equations (6) and (7) can be re-written by grouping the measurements as node wise clusters, which is shown below.

All possible measurements at nth node cluster is

$$\left[\Delta z^n\right] = [\Delta P_n, \Delta Q_n, \Delta p_{nk}, \Delta q_{nk}, \Delta v_n, \Delta \delta_n]; \tag{8}$$

$$\left[\mathrm{A_n^P}\right] = \left\{[J_n^P]^T * [W_{Pn}] * [J_n^P]\right\}; \quad \left[\mathrm{A_{nk}^q}\right] = \sum_{\mathrm{k}}^{\mathrm{cb}} \left\{[J_n^q]^T * [W_{qn}] * [J_n^q]\right\} \tag{9}$$

$$\left[\mathrm{b_n^P}\right] = \left\{[J_n^P]^T * [W_{Pn}] * [\Delta P_n]\right\}; \quad \left[\mathrm{b_{nk}^q}\right] = \sum_{\mathrm{k}}^{\mathrm{cb}} \left\{[J_n^P]^T * [W_{Pn}] * [\Delta q_{nk}]\right\}, \tag{10}$$

similarly for other measurements.

where 'cb' number of connected nodes of 'nth' bus; 'k' varies up to 'cb'.

$$[{}^nA] = [A_n^P + A_n^Q + A_{nk}^p + A_{nk}^q + A_n^v + A_n^\delta] \tag{11}$$

$$[{}^nb] = [b_n^P + b_n^Q + b_{nk}^p + b_{nk}^q + b_n^v + b_n^\delta] \tag{12}$$

$$[\mathrm{A}] = \sum_{\mathrm{i=1}}^{\mathrm{n}} \mathrm{A^i} \text{ and } [\mathrm{b}] = \sum_{\mathrm{i=1}}^{\mathrm{n}} \mathrm{b^i} \tag{13}$$

Note: - $A^{\mathrm{P/Q/p/q/v/}\delta}$ - stands for type of measurements

n - stands for node

The equation (13) or (6) and (7) can be used to compute Matrices 'A' and 'b'

2.3 Jacobian Distributed Computing:

Step 1: - Read Input data

```
Step 2: - From system parameters find Ybus.
Step 3: - Initialize $[vi]^T = 1$; and $[\partial i]^T=0$;
 For( r = 1 to m)
{ Step 4.: - Find $z_r^{cal}$
Step 5: - Find $J_r$ : - Jacobian row corresponding ‘$z_r$ ‘ and $\Delta z_r = z_r^{mes} - z_r^{cal}$
Step 6: - Find $J_r^T * W_{rr} * J_r = [A] + [A_r]$ and $J_r^T * W_{rr} * \Delta z_r = [b] + [b_r]$    }
Step 7: - Find $[\Delta x_i] = [A]^{-1} * [b]$;
Step 8: - check for $[\Delta x_i]$
 if $[\Delta x_i] << €$;
if No - Update [xi]; $[xi]^{new} = [x_i]^{old} + [\Delta x_i]^{old}$; then repeat from step 6.
if yes – Stop
```

2.4 Advantages: - From the above facts it is clear that

i. The task from Step 1 to step 4 can be distributed to ‘n’ processors.
ii. Dynamic size of Jacobian is (1 * 2n-1) but in conventional method ‘J’ size is (m * 2n-1)
iii. Dynamic size of Δz is just a simple variable. In conventional method ‘Δz’ size is (m*1).
iv. No need to store the z^{cal} in file/array variables.
v. Local indexing for ‘Jacobian’ elements is easier
vi. Multiplication of $J_r^T * W_{rr} * J_r = [A] + [A_r]$ and $J_r^T * W_{rr} * \Delta z_r = [b] + [b_r]$ can be done easily for non zero elements using local indexing.
vii. It is observed that the multiplication time required by the new method is much lesser than the conventional method.

2.5 Sparse Technique

The following C++ program demonstrates the use of new logic, taking line flow measurement example. Let the global variable sij[][] and h[] structure is

- sij[i][1] na; sij[i][2]- node from (i); sij[i][3]- node to (j)
- sij[i][4] = pij measured & sij[i][5] = qij measured
- sij[i][6] = y/2 half line charging between i & j
- sij[i][7] = Gij & sij[i][8] = Bij
- h[] = Jacobian row variables

find_jaco_lf(int r,int n): - is the sub-program to find the elements of A and b corresponding to a measured line flow values using this new technique. In the following program all the computations are focused only for non-zero elements.

```
void find_jaco_lf(int r,int n)
 {
  int i,j,q,p,kl,g,k,l,x1,x2,x3,x4; // (x1 to x4) non-zero location of Jacobian row
  long double delz,a1,b1,deg,yij,tea, smp,smq;
  for(q=0;q<=2*n;q++)
       h[q].x = 0.0;
  for(g=1;g<=kl;g++) // kl - number of line flow measurements
      {
              i = (sij[g][2].x); // node 1 is taken as reference
              j = (sij[g][3].x);

              x1=i-1;
              x2=j-1;
              x3=n+i-1;
              x4=n+j-1;
              z[1].x=x1;
              z[2].x=x2;
              z[3].x=x3;
              z[4].x=x4;
                          a1 = sij[g][7].x;
                          b1 = sij[g][8].x;
                          yij = sqrt((a1*a1)+(b1*b1));
                          tea= atan(b1/a1);
                          if(a1==0.0)
                          tea=-(11./7.);
                          deg = (del[i].x - del[j].x - tea);
              smp=yij*(v[i].x*v[i].x*cos(tea)-(v[i].x*v[j].x*cos(deg)));
              smq=yij*(v[i].x*v[i].x*sin(-tea)- v[i].x*v[j].x*sin(deg)) -  (sij[p][6].x* v[i].x * v[i].x);
/*----To find  Jr- for line flow "pij. For the sake of simplicity ‘Wii’ is assumed to be unity*/
             h[x1].x = v[i].x*v[j].x*yij*sin(deg);                                /*h5i*/
             h[x2].x= -v[i].x*v[j].x*yij*sin(deg);                                /*h5j*/
             h[x3].x=yij*(2*v[i].x*cos(tea)-v[j].x*cos(deg));/*h6i*/
             h[x4].x = -v[i].x*yij*cos(deg);                                      /*h6j*/
/*----A = A + Jr^T*Jr and b = b+ Jr^T*Δzr---- (for line flow "pij")*/
 delz = (sij[g][4].x-smp);
              for(p=1;p<=4;p++)
                         {
               k=z[p].x;
               b[k].x = (b[k].x + h[k].x*delz);
               for(q=1;q<=4;q++)
                          {
                          l=z[q].x;
                          a[k][l].x = (a[k][l].x + h[k].x*h[l].x);
                                          } }
/*----To find  Jr- for line flow "qij"…..*/
            h[x1].x = -v[i].x*v[j].x*yij*cos(deg);                               /*h7i*/
            h[x2].x =  v[i].x*v[j].x*yij*cos(deg);                       /*h7j*/
            h[x3].x = (2*v[i].x*sin(-tea)-v[j].x*sin(deg))*yij;                  /*h8i*/
            h[x4].x = -v[i].x*yij*sin(deg);                                      /*h8j*/
  /*----A = A + J_r^T*J_r-and b = b+ J_r^T*Δz_r---- (for line flow "qij")*/
              delz = (sij[g][5].x-smq);
              for(p=1;p<=4;p++)
                         {
               k=z[p].x;
               b[k].x = b[k].x + h[k].x*delz;
               for(q=1;q<=4;q++)
                         {
               l=z[q].x;
               a[k][l].x = a[k][l].x + h[k].x*h[l].x;
                      } } } }
```

2.6 Computational Efforts

A matrix is said to be sparse if a given finite discrete ample space Ω and a non-empty set of sample S is such that the cardinality |S| of S is small compared to the cardinality | Ω | of Ω i.e. |S| << | Ω |. As discussed L.P.Singh [2],

2.6.1 Effort for Conventional technique

- Let p= |S| and Q = | Ω |.
- 't_i' is the time taken to perform operation by elements 'i' of 'S'
- 'r_i' is the additional time taken to retrieve an element of elements 'S' in a compact storage scheme.
- 's_i' is the additional time taken to store an element of elements 'S' in a compact storage scheme.

The total processing time without compact storage is = $\sum_{i=1}^{Q} t_i$

The total processing time with compact storage is = $\sum_{i=1}^{p} t_i + r_i + s_i$

For sparse matrix $\sum_{i=1}^{p} t_i + r_i + s_i < \sum_{i=1}^{Q} t_i$

2.6.2 Effort for New technique: -

As seen earlier no special storing and retrieving scheme is required for Jacobian in this new technique,

The total processing time for new scheme = $\sum_{i=1}^{p} t_i$

Normally J^T*Wi*J is carried out by taking the row of J^T into column of J. In conventional method, the identification of non-zero elements of each column of J takes more time than row wise non-zero elements identification of J. This is because of the fact that the non-zero elements in each row of Jacobian have direct relation with the corresponding row of network incident matrix.

The time 'r_i' in the new technique will be very small as compared to the old scheme because of local dynamic indexing. The time 'S_i' in the new technique is considered zero without compact storage for the resultant matrix 'A'. When the measurements are grouped node-wise the resultant matrix 'A' for the given node will be dense matrix.

3 Example & Results

A simple four bus example has been considered. First 'A' is calculated using conventional method, where A= (J^T*W*J). Next 'A' is computed using new method, where A= $\sum_{j=1}^{m}(J_j^T * W_{jj} * J_j)$; 'm' is the total number of measurements taken.

The new method (Distributed technique) results are also processed with single processor.

3.1: Circuit & Input tables

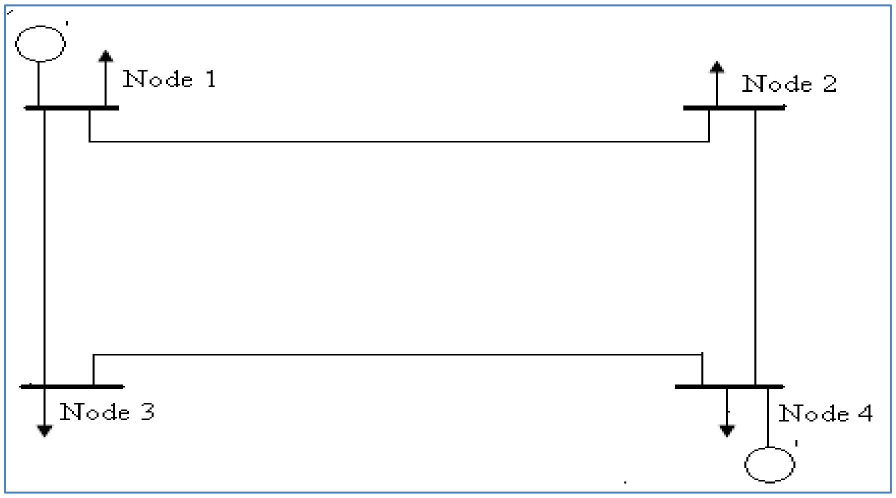


Fig-1: - Circuit diagram

Table 1: Line data

I - j	R	X	G	B	Y/2
1--2	0.01008	0.0504	3.815629	-19.0781	0.05125
1--3	0.00744	0.0372	5.169561	-25.8478	0.03875
2--4	0.00744	0.0372	5.169561	-25.8478	0.03875
3--4	0.01272	0.0636	3.023705	-15.1185	0.06375

Table 2: Injected power, Voltage & Phase measurements

P_i	Q_i	V_i	∂_i
1.3718	0.8431	1	0
-1.6945	-1.0735	0.982	-0.0171
-1.9914	-1.2416	0.969	-0.032
2.3809	1.3299	1.02	0.026

Table 3 Line flow measurements

na	i	j	P_{ij}^{mes}	Q_{ij}^{mes}
1	1	2	0.3877	0.2825
1	1	3	0.9792	0.6514
2	2	1	-0.3852	-0.2809
2	2	4	-1.3138	-0.715
3	3	1	-0.969	-0.5999
3	3	4	-1.0266	-0.5446
4	4	2	1.3311	0.8013
4	4	3	1.0266	0.6361

3.2 Results:

The results are divided into two parts.

Part-I: - Tables R1 to R6 shows the resultant matrix 'A' by either method for first iteration.

(Note: - In the below tables "the smallest value is represented by 'E-06')

Table-R1: - $[A]=[J^T * W * J]$ size of 'A' is $=[(2n-1)\ x (2n-1)]$ and size of 'J' is $=[m\ x\ (2n-1)]$

5320.17	919.265	-3698.6	1.56322	-3.68118	E-06	1.71871
919.265	4544	-1763.67	2.11791	E-06	-2.72392	1.00527
-3698.6	-1763.67	4544.01	E-06	2.11791	1.00527	-2.72398
1.56322	2.11791	E-06	5304.04	-2532.97	-3688.65	919.265
-3.68118	E-06	2.11791	-2532.97	5304.04	919.265	-3688.65
E-06	-2.72392	1.00527	-3688.65	919.265	4527.25	-1757.47
1.71871	1.00527	-2.72398	919.265	-3688.65	-1757.47	4527.25

Table-R2: - $[A]=\sum_{i=1}^{n}\left[{}^{i}A\right]$ size of 'A' is $=[(2n-1)\ x (2n-1)]$ and size of $'J_r'$ is $=[1\ x\ (2n-1)]$

5320.17	919.265	-3698.6	1.56321	-3.6811	E-06	1.71868
919.265	4544.01	-1763.67	2.11794	E-06	-2.72402	1.0053
-3698.6	-1763.67	4544.01	E-06	2.11794	1.0053	-2.72402
1.56321	2.11794	E-06	5304.04	-2532.97	-3688.65	919.265
-3.6811	E-06	2.11794	-2532.97	5304.04	919.265	-3688.65
-E-06	-2.72402	1.0053	-3688.65	919.265	4527.25	-1757.47
1.71868	1.0053	-2.72402	919.265	-3688.65	-1757.47	4527.25

Table-R3: - $[{}^{1}A]=[A_1^P + A_1^Q + A_{1k}^p + A_{1k}^q + A_1^v + A_1^\delta]$

757.068	512.853	**0**	1.56323	E-06	E-06	**0**
512.853	1389.67	**0**	2.11793	E-06	E-06	**0**
0	**0**	**0**	**0**	**0**	**0**	**0**
1.56323	2.11793	**0**	3157.3	-1266.49	-1897.87	**0**
E-06	E-06	**0**	-1266.49	757.068	512.853	**0**
E-06	E-06	**0**	-1897.87	512.853	1389.67	**0**
0	**0**	**0**	**0**	**0**	**0**	**0**

Table-R4: - $[{}^{2}A] = [A_2^P + A_2^Q + A_{2k}^p + A_{2k}^q + A_2^v + A_2^\delta]$

3173.44	**0**	-1902.52	E-06	-3.6811	**0**	E-06
0	**0**	**0**	**0**	**0**	**0**	**0**
-1902.52	**0**	1389.67	E-06	2.11793	**0**	E-06
E-06	**0**	E-06	757.068	-1266.49	**0**	512.853
-3.6811	**0**	2.11793	-1266.49	3157.3	**0**	-1897.87
0	**0**	**0**	**0**	**0**	**0**	**0**
E-06	**0**	E-06	512.853	-1897.87	**0**	1389.67

Table-R5: - $[{}^{3}A] = [A_3^P + A_3^Q + A_{3k}^p + A_{3k}^q + A_3^v + A_3^\delta]$

0	**0**	**0**	**0**	**0**	**0**	**0**
0	2678.92	-881.837	E-06	**0**	-2.72402	E-06
0	-881.837	475.425	E-06	**0**	1.00529	E-06
0	E-06	E-06	1389.67	**0**	-1790.78	406.412
0	**0**	**0**	**0**	**0**	**0**	**0**
0	-2.72402	1.00529	-1790.78	**0**	2662.16	-878.737
0	E-06	E-06	406.412	**0**	-878.737	475.425

Table-R6: - $[{}^{4}A] = [A_4^P + A_4^Q + A_{4k}^p + A_{4k}^q + A_4^v + A_4^\delta]$

1389.67	406.412	-1796.08	**0**	E-06	E-06	1.71871
406.412	475.424	-881.836	**0**	E-06	E-06	1.00529
-1796.08	-881.836	2678.92	**0**	E-06	E-06	-2.72402
0	**0**	**0**	**0**	**0**	**0**	**0**
E-06	E-06	E-06	**0**	1389.67	406.412	-1790.78
E-06	E-06	E-06	**0**	406.412	475.424	-878.737
1.71871	1.00529	-2.72402	**0**	-1790.78	-878.737	2662.16

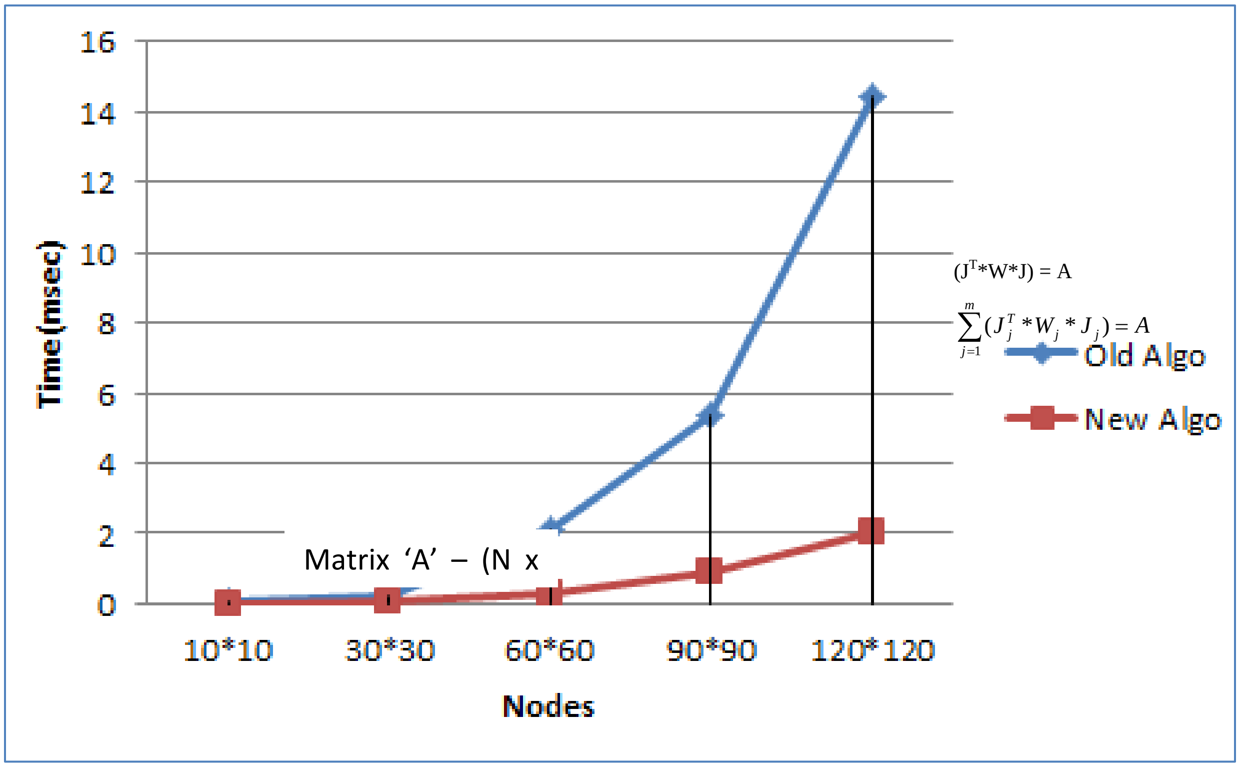
Part-II: -

The graph in figure 2 shows the computing time for 'A' by either method.

In the graph the New Algorithm and Old Algorithm represents the computational time for the new and old method respectively. The above relation is obtained under following assumption.

- The number of measurements is assumed to be equal to the number of state variables.
- Approximately 15% of the Jacobian elements are assumed to be non-zero; this fact is based on the practical system.

The profiling is conducted on Core2-Duo X-86 2GB RAM machine with Operating System Windows 7, we were able to fetch the above results. The result may vary with the different hardware configuration but the ratio of the Old Vs New Algorithm remains almost constant no matter what hardware configuration has been adapted.



(Note: - Both the above results are obtained using single processor.)

Figure 2: Computation time

4 Conclusions

The results show that the resultant matrixes 'A' computed by both methods are same. In Old Algorithm (Algo) the time was exponentially increasing with the number of network nodes, whereas in the new Algo the exponential growth has been reduced to almost linear growth. Using new algorithm, the time can be reduced further by employing more number of processors. Even though both the methods are mathematically same, new method results in drastic reduction in computational time. In the new method up to Jacobian product can be computed by parallel processing using the equations (6) and (7). By grouping the measurements in the form of node clusters [equations (8), (9) & (10)] it is possible to divide the Matrix 'A' (& b) as node cluster wise gain matrix [nA] and [nb]. The size of [^{n}A] and [^{n}b] is directly related to the size of node cluster state variables. For example, at node cluster No. 2, [refer table-R4] state variables are (∂2, ∂4,v1, v2,v4) taking node-1 as angle reference node. Hence the size of [^{2}A] is (5x5) and not (7x7) similarly, the size of [^{2}b] is (5x1) & [^{2}Δx] is (5x1). It is not necessary to have separate reference node at each node cluster. Hence, solution for [Δxi] can also be divided into 'n' independent sub tasks which will further reduces the computational time, which will be presented in further papers.

ACKNOWLEDGEMENT

The authors would like to thank Mr. Antriksha Somani, Assit. Prof., Dept. Cs, MIT, Indore, MP, for his help in programming and profiling.

REFERENCES

[1]. H.N. Udupa, Dr. H.R.. Kamath et al., Modified electric power system state estimation – Multi- processing technique, IMPACT: International Journal of Research in Engineering & Technology (IMPACT: IJRET) ISSN 2321-8843 Vol. 1, Issue 5, Oct 2013, 47-56

[2] L.P. Singh, third edition 1992, *"Advanced Power System Analysis & Dynamic,"* New Age International Publishers. P.No. 254 – 287.

[3] L.P Singh and H.C Srivastava. 'Sparsity and Optimal ordering'. Journal of The Institute of Engineers (India), vol.57,pt EL6, June 1977, p274

[4] K.K.Goyal and L.P Singh. 'Optimal Elimination of Sparse System Using Dynamic Programming Technique'. Proceeding CS 9-81, March 1-4,1981, New Delhi.

[5] EPRI, "Exploring applications of Digital parallel Processing to Power System Problems," Seminar proceedings, Oct 4- 7, 1979.

[6] Y. Wallach, E. Handschin, C.Bongers, "An Efficient Parallel Processing Method for Power system State Estimation," Trans. IEEE, Vol. PAS-100, Nov.1981, pp.4402 – 6.

[7] M.Y.Patel, A.A.Girgis,, "Two-Level State Estimation for Multi-Area Power system", 1-4244-1298/$25.00 @2007IEEE

[8] Patel M. Y., Girgis A. A., "Two-Level State Estimation for Multi-Area Power System",1-4244-1298-6/07/$25.00 ©2007 IEEE.

BIOGRAPHIES

First Author was born in south Indian Village, Udupi District in 1963. He received the B.E in Electrical Engineering from National Institute of Technology (formerly known as "Regional Engineering College"), Silchar, Gwhati University of India, in 1988 and M.Tech honors degree, in Power System from IIT Roorkee (formerly known as "University of Roorkee"), Utarakhand, India, in 1995. The Author is the Ph.D scholar of Mewar University, Rajastan, India. From 1990 to 1999, he was a faculty at Manipal Institute of Technology, Manipal, India. From 2000 to 2009 he has been an Associate Professor with Sikkim-Manipal University.

Dr. H Ravishankar Kamath, completed his Bachelor of Engineering from Mysore University in the year 1989 and Master of Technology from National Institute of Technology Surathkal Karnataka in the year 1996 in power and energy system. He has done PhD from Manipal University in the year 2008.He has subject specification in soft computing and its various application in the field of power system and Nonconventional energy systems