An Efficient Parallel Gauss-Seidel Algorithm for the Solution of Load Flow Problems

Raed Alqadi[1] and Maher Khammash[2]
[1]Department of Computer Engineering, An-Najah National University, Palestine
[2]Department of Electrical Engineering, An-Najah National University, Palestine

Abstract: *In this paper, a parallel algorithm for solving the load flow problem of large power systems is presented. This algorithm uses a parallel virtual machine implemented as a distributed system built from readily available PCs. The proposed algorithm is based on the Gauss-Seidel algorithm usually used in the solution of load flow problems. This algorithm is parallelized by distributing the bus voltages among a set of processors such that each processor will be working on 1/n of the bus voltages in the circuit, where n is the number of processors. Since it is virtually impossible to obtain a parallel processing machine in our country, the algorithm is developed over a distributed system which consists of a network of PCs. Even though the communication overhead is much more than that in a real parallel machine, the results show that large power systems can be solved in much less time compared to the time required for sequential algorithm usually used, and that with proper selection of the number of processors, the execution time is reduced by almost a factor of the number of processors.*

Keywords: *Parallel algorithm, distributed system, load flow, communication overhead, execution time.*

Received November 16, 2005; accepted April 22, 2006

1. Introduction

Load flow studies are of great importance in planning and designing the future expansion of power systems as well as in optimizing existing systems for best performance. The principal information obtained from a power-flow study consists of the magnitude and phase angle of the voltage at each bus and the real and reactive power flowing in each line.

Traditional solutions of the load-flow problems follow an iterative process by assigning estimated values to the unknown bus voltages and calculating a new value for each bus voltage from the estimated values at the other buses, the real power specified, and the specified reactive power or voltage magnitude. A new set of values for voltage is thus obtained for each bus and still used to calculate another set of bus voltages in a sequential algorithm. The iterative process is repeated until the changes at each bus are less than a specified tolerance value.

The sequential algorithm usually used in load flow calculations may result in large number of iterations over a large set of data and thus requires long execution time. Over the past two decades, there has been a great interest in using parallel and distributed computer systems in the computation as well as monitoring of large power systems' performance and parameters such as node voltages, currents, and power flow, which has been applied in this study. Examples of such studies can be found in [3, 5, 7, 8, 9, 11]. In this paper, we present a parallel technique based on distributing the computation over a set of computers interconnected with a regular Local Area network (LAN). The technique mimics parallel processing through a distributed system thus creating a parallel virtual machine. Parallel virtual machines have been proposed by many researchers; see for example [2, 6, 10, 12, 13]. We implemented our own machine because it has been optimized for solving power systems; mainly parallelizing the well-known Gauss-Seidel algorithm which is commonly used in load flow analysis of power systems.

The aim of this study is to create a parallel algorithm using parallel processing to solve the problem of load flow by Gauss-Seidel method. Despite the fact that Gauss-Seidel algorithm is inherently serial, it is possible to gain performance by applying parallel techniques [7]. In [1], a parallel Gauss-Seidel algorithm that targets multi-grid systems and runs on distributed memory computers is proposed. In [7], a technique has been proposed to parallelize Gauss-Seidel algorithm by performing specialized orderings on sparse matrices. It is shown in [5] that it is possible to eliminate much of the data dependencies caused by precedence in the calculations. In this paper, we present a much simpler parallel algorithm that achieves high performance by distributing the calculations of the bus voltages among several processors. In our technique, each processor is responsible for calculating a subset of bus voltages. This parallel algorithm has been examined on several power systems to ensure its capability and effect upon reducing the computer execution time in comparison with the sequential algorithm usually used. The algorithm was evaluated

using a large number of power networks generated by a random network generator program written by the authors.

The rest of this paper is organized as follows. Section 2 provides an overview of the Gauss-Seidel load flow algorithm. Section 3 presents the proposed parallel version of the Gauss-Seidel algorithm. Section 4 presents the results of the proposed approach including a comparison to the sequential algorithm. Finally, conclusions are presented in section 5.

2. Gauss-Seidel Method

The Gauss-Seidel method is an iterative process which starts by assigning estimated values to the unknown bus voltages. Using the estimated bus voltages and the specified real and imaginary power values, a new value for each bus voltage is calculated at the end of each iteration. The process is repeated until the difference between each bus voltage and its corresponding value in two successive iterations is less than a predefined tolerance value.

For a total of *N* buses, the calculated voltage at any bus *K* is expressed as a function of the real and reactive power delivered to a bus from generators or supplied to the load connected to the bus, the estimated or previously calculated voltages at the other buses, and the self- and mutual admittances of the nodes as given in equation (1) [4].

$$V_k^{(i)} = \frac{1}{Y_{kk}} \left[\frac{P_k - jQ_k}{V_k^{(i-1)}} - \sum_{j=1}^{k-1} Y_{kj} V_j^{(i)} - \sum_{j=k+1}^{N} Y_{kj} V_j^{(i-1)} \right] \quad (1)$$

Where:

V_k : Calculated voltage of bus *k*.

Y_{kk} : Self admittance of bus *k*.

Y_{kj} : Mutual admittance between buses *k* and j.

P_k: and Q_k: Scheduled real and reactive power entering the system at *bus k.*

V_j: Most recently calculated values for the corresponding buses or the estimated voltage if no iteration has yet been made at that particular bus.

(i) and (i - 1) denote current and previous iteration, respectively.

In the original sequential algorithm of Gauss-Seidel method, the voltage at each bus at any iteration is calculated by using the voltages of previous buses calculated at the same iteration and the voltages of the next buses calculated at the previous iteration as shown in equation (1). The entire process is carried out again and again until the amount of correction in voltage at every bus is less than some predetermined precision index.

Experience with the Gauss-Seidel method in solving large power flow problems has shown that an excessive number of iterations are required before the voltage corrections are within an acceptable precision index if the corrected voltage at a bus merely replaces the best previous value as the computations proceed from bus to bus which results in long computer time.

In this study, it is shown that the total execution time is reduced considerably by using parallel processing for the solution of load flow problem since each processor (computer) is only responsible for a subset of bus voltages and calculation of the voltages in done concurrently. In the next section, we present our version of the parallel Gauss-Seidel algorithm.

3. Parallel Algorithm

The idea of using parallel processing in the solution of load flow problem by Gauss-Seidel method starts with dividing the whole group of bus voltages into subgroups where the calculation of the bus voltages of these subgroups is performed by several processors (clients) concurrently. Since the parallel processing is emulated by a distributed system composed of PCs interconnected via LAN, from now on we will refer to the processors or PCs by the term clients and server. The server will be responsible for coordinating the distribution and execution of the algorithm while the clients will be responsible for the execution of the algorithm. The calculation of the bus voltages is distributed among several clients such that each client is responsible for the calculation of the bus voltages of the subgroup to which it belongs, in this way each client calculates the bus voltages of its subgroup independently of other clients. The clients used in the implementation are heterogeneous PCs deploying Microsoft Windows XP.

Each client calculates the bus voltages of its subgroup according to equation 2 in the following manner. If bus j does not belong to the same subgroup, the value of V_j is substituted from the previous iteration, but if bus j belongs to the same subgroup then the value of V_j is substituted from the same iteration if j < k or and from the previous iteration if j > k. The parallel algorithm can be presented mathematically as shown in Figure 1. Also, a pictorial representation of the algorithm is shown in Figure 2. In this figure, 12 buses are distributed over 4 clients (computers) where each client computes the voltages for 3 buses.

3.1. Implementation

The parallel algorithm has been implemented using a network of 20 computers connected over LAN; the program was written in C# and used TCP/IP sockets for communication. The following paragraphs describe the main steps of the algorithm.

3.2. Initial Step

- The Server distributes the network definitions to the clients.

- From the clients' number connected, the server calculates the portion of data each client must work on. For example, if the number of clients is 8 and the number of the bus voltages in the network is 2000 bus, then the number of bus voltages each client has to work on is 2000/8 = 250 bus voltages.
- Every client is sent the complete set of data including the following:
  - The set of the initial estimated values of the bus voltages in the network.
  - The node admittance matrix of the network that includes the self-and mutual admittances.
  - The scheduled real and reactive power entering the system at each bus.
  - The range of the elements the client is to work on. For example 8 clients work over 2000 elements, then the first client will work on the range of elements form 1-250, while the second will work over 251-500, and so on.
- The server sends the "*start execution*" message to the clients.

Parallel Gauss Seidel Algorithm
Let $C = \{c_1, c_2, \ldots c_m\}$ *be the set of available clients (computers) where m is the number of computers, and let N be the number of buses*
Client L computes the voltages $\{V_x, \ldots, V_y\}$ *where*

$$x = \left\lceil \frac{N}{m} \right\rceil * (L-1) + 1 \quad and \quad y = Min\left(\left\lceil \frac{N}{m} \right\rceil * L, N \right)$$

For each k in {x, ..., y) compute

$$V_k^{(i)} = \frac{1}{Y_{kk}} \left[\frac{P_k - jQ_k}{V_k^{(i-1)}} - \sum_{j=1}^{x-1} Y_{kj} V_j^{(i-1)} - \sum_{j=x}^{k-1} Y_{kj} V_j^{(i)} - \sum_{j=k+1}^{N} Y_{kj} V_j^{(i-1)} \right] \quad (2)$$

Broadcast the vector {Vx, Vx + 1, ..., Vy}
Obtain the voltages {V1, ..., VN} - {Vx, ..., Vy} from the other clients

Figure 1. Parallel Gauss-Seidel algorithm.

3.3. Iterative Steps

- Every client calculates the values of the bus voltages to which it is dedicated.
- When a client finishes iteration over its range of data, it broadcasts the vector of node voltages it computed with a tag identifying its identity number. Afterwards, the client sleeps waiting for a notification from the server to start a new iteration.
- When a client receives the computed bus voltages from all other clients, it replaces the old bus voltages with the new received values.
- When all clients send their vectors of node voltages, the server sends a "continuation message" to alert the clients to start a new iteration.
- The previous iterative steps are repeated until all clients converge, or until a certain predefined number of iterations end with no convergence.
- When the convergence is reached by the clients, each client sends a completion message to server. When this message is received from all clients, the server displays the results. At this moment, the execution is completed.

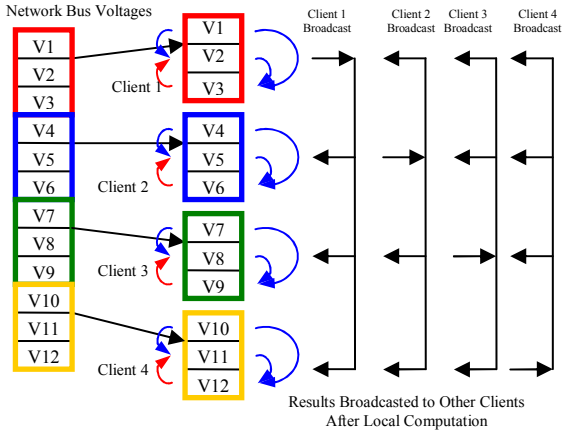

Figure 2. Distribution of the calculation of bus voltages among several clients.

4. Results and Discussion

To evaluate the algorithm, the parallel algorithm was compared to the sequential algorithm. To achieve this, a random network generator was developed in order to provide test power networks. Several experiments were conducted using different number of clients and different size of networks. Generally, for large networks (1000 nodes or more) the algorithm produced significant speed up. Of course this is the desired behavior since the algorithm is intended for large networks.

The amount of communication overhead becomes insignificant as compared to the processing time even though it is based on LAN communication. But for small networks (200 nodes or less) the amount of communication overhead is significant and hence the serial algorithm is superior.

Figure 3 shows the execution time in seconds for networks of 1000 buses versus the number of computers. It is clear that the execution time decreases significantly as the number of computers is increased. But as the number increases beyond a certain limit, the communication time becomes significant and thus the increase becomes marginal. However, the figure clearly shows that our algorithm is very efficient in reducing the execution time for large network using a reasonable number of computers, e. g., 5-20.

Figure 4 shows the speedup versus the number of clients (computers), the results were calculated by dividing the sequential program execution time by the execution time of the parallel algorithm for each respective number of clients. The results in Figures 3 and 4 were obtained by taking the average execution time of 10 randomly generated networks; each is a 1000 bus network.

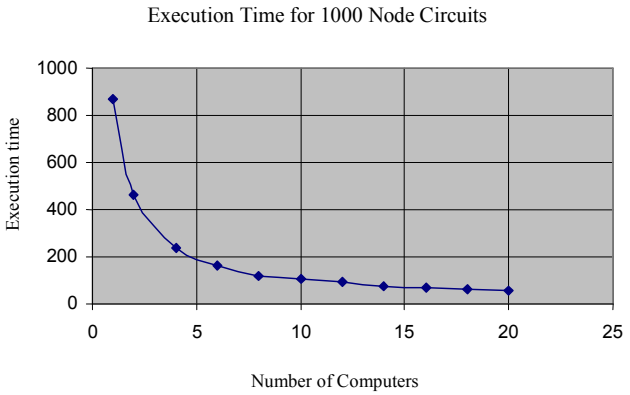

Figure 3. Execution time (s) versus the number of clients for 1000 bus networks.

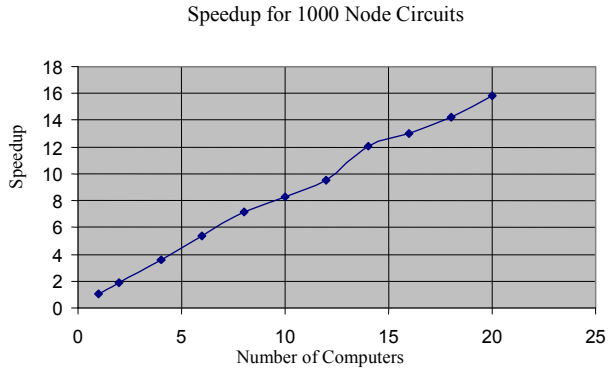

Figure 4. Speedup versus number of clients for 1000 bus networks.

Figure 5 and 6, respectively, show the execution time and the speedup for networks of 500 buses versus the number of computers. It can be seen that as the number of computers is increased beyond 12, the execution time starts to increase. Such result is expected because the number of buses in the power networks is relatively small (500) and hence a small number of computers (less than 10) will be needed. Increasing the number beyond that value will result in high communication overhead compared to the computation time of the bus voltages. Consequently, the suggested algorithm is effective for networks with a medium number of buses between 200 and 1000 when using a reasonable number of clients in order not to increase the communication overhead as compared to the computation time of the bus voltages.

From studying these results, it is clear that for large power systems the suggested algorithm is effective. This algorithm reduced the execution time considerably compared with the sequential algorithm usually used. The amount of communication overhead becomes insignificant as compared to the processing time when the size of the power circuit is considerably large and a reasonable number of computers, e. g., 5-20, is used.

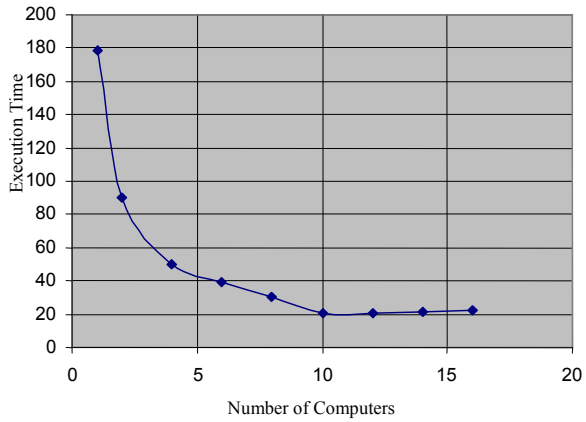

Figure 5. Execution time (s) versus the number of clients for 500 bus networks.

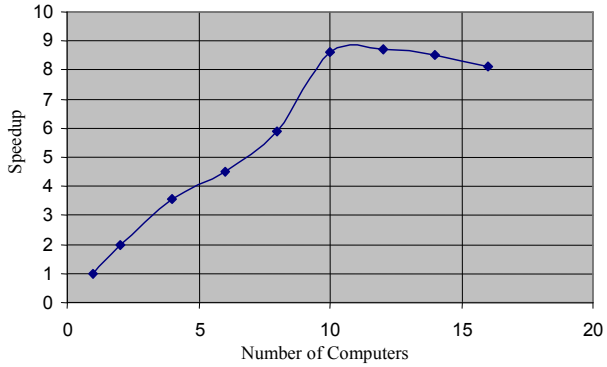

Figure 6. Speedup versus the number of clients for 500 bus networks.

5. Conclusion

In this study, a parallel algorithm of load flow calculation using Gauss-Seidel method is realized. This algorithm is based on the distribution of the calculations of the bus voltages among several clients operating in parallel. The parallel processing is emulated by a distributed system composed of PCs interconnected via LAN. The proposed algorithm was tested on several test power networks and compared with the sequential algorithm usually used.

The obtained results show that the execution time of the parallel algorithm has been considerably reduced in comparison with the execution time of the sequential algorithm for power networks of large number of buses. The amount of communication overhead becomes insignificant as compared to the processing time even though it is based on LAN communication.

The proposed parallel algorithm which has been implemented using distributed processing is an efficient algorithm and is recommended by the authors in the calculation of load flow for the networks with large number of buses.

References

[1] Adams M. F., "A Distributed Memory Unstructured Gauss-Seidel Algorithm for Multi-Grid Smoothers," *in ACM/IEEE Proceedings of SC01 Performance Networking and Computing*, 2001.

[2] Al Geist, Beguelin A., Dongarra J., Jiang W., Manchek R., and Sunderam V. S., *PVM: Parallel Virtual Machine: A Users' Guide and Tutorial for Networked Parallel Computing,* MIT Press, 1994.

[3] Bruggencate M. and Chalasani S., "Parallel Implementations of the Power System Transient Stability Problem on Clusters of Workstations," *in Proceedings of Supercomputing*, San Diago, CA, 1995.

[4] Grainger J. G. and Stevenson W. D., *Power System Analysis*, McGraw Hill, New York, 1994.

[5] Gupta A., "Recent Advances in Direct Methods for Solving Unsymmetric Sparse Systems of Linear Equations," *ACM Transactions on Mathematic Software*, vol. 28, no. 3, pp. 301-324, 2002.

[6] Hariri S. and Parashar M., *Tools and Environments for Parallel and Distributed Computing*, Wiley-IEEE, 2004.

[7] Koester D. P., Ranka S., and Fox G. C., "A Parallel Gauss-Seidel Algorithm for Sparse Power System Matrices," *in Proceedings of Supercomputing*, pp. 184-193, 1994.

[8] Koester D. P., Ranka S., and Fox. G. C., "Parallel Block-Diagonal-Bordered Sparse Linear Solvers for Electrical Power System Applications," *in Proceeding of the Scalable Parallel Libraries Conference*, IEEE Press, 1994.

[9] Mahesh J., Karypis G., Kumar V., Gupta A., and Gustavson F. G., "An Efficient and Scalable Parallel Sparse Direct Solver," *in Proceedings of the Ninth SIAM Conference on Parallel Processing for Scientific Computing*, March 1999.

[10] Tylavsky D. J. and Bose A., "Parallel Processing in Power Systems Computation," *IEEE Transactions on Power Systems*, vol. 7, no. 2, pp. 629-638, 1992.

[11] Wei-keng L., Coloma K., Choudhary A., and Ward L., "Cooperative Write-Behind Data Buffering for MPI I/O," *in the Proceedings of 12th Euro PVM/MPI Conference*, pp. 102-109, 2005.

[12] "PVM: Parallel Virtual Machine," available at: http://www.csm.ornl.gov/pvm/.

[13] Wikipedia, "Parallel Virtual Machine," *The free Encyclopedia*, available at: http://en.wikipedia.org/wiki/PVM.

Raed Alqadi received his MSc and PhD in computer engineering from the University of Wisconsin-Madison, USA in 1995. He joined the Electrical Engineering Department in 1996 and founded the Computer Engineering Department in 2000. Currently, he is a staff member at the Computer Engineering Department at An-Najah National University. Previously, he was the chair of Electrical Engineering, Computer Engineering Department, and dean of the IT College. His research interests include real time systems, computer architecture, embedded microcontrollers, and web-programming.

Maher Khammash received his MSc and PhD degrees in electrical engineering from Moscow Power Engineering Institute, Technical University, Moscow, Russia, in 1989 and 1993, respectively. He joined An-Najah National University, Nablus, Palestine, in 1993 where he is currently an assistant professor at the Department of Electrical Engineering. He has taught in the areas of electric power systems analysis and planning, CAD applications in electric power systems, and electrical machines. His research interests are computer aided design in electric power systems, optimum operation of electrical networks, and reactive power compensation in distribution and transmission networks.