Proceedings of the Eighth IASTED International Conference
POWER AND ENERGY SYSTEMS
October 24-26, 2005, Marina del Rey, CA, USA

POWER FLOW ANALYSIS ON AN FPGA-BASED VECTOR COMPUTER

Muhammad Z. Hasan[1], Sotirios G. Ziavras[1,2] and Tae-Gyu Chang[2]
[1]Electrical and Computer Engineering Department
New Jersey Institute of Technology, Newark, NJ 07102, U.S.A.
[2]School of Electrical and Electronics Engineering
Chung-Ang University, 156-756, Seoul, Korea

ABSTRACT
The solution to a set of linear equations given in the form Ax = b, where A is an n×n sparse matrix and b is an n-element vector, can be obtained with the W-matrix method for power flow studies. The characteristics of this method are explored here for sparse linear systems present in such studies and an enhanced vector processor is proposed to support them directly in hardware. The effects of customized instructions, instruction chaining, and matrix density are evaluated. The impact of multiple pipelined functional units, multiple data buses, and vector register size is analyzed as well. Our implementation of the vector processor on an FPGA (Field-Programmable Gate Array) is discussed and benchmark results are presented.

KEY WORDS
Power flow, sparse matrix, vector computer, FPGA

1. Introduction

Many power flow problems need to solve a set of linear equations [1]. A direct method can solve Ax = b by first factorizing the sparse matrix A into the lower-diagonal L, diagonal D, and upper-diagonal U components [2]. The inverse factor (or W-matrix) method is often used to solve Ax = b, where the inverses of L and U are computed in a very efficient manner [2, 3]. This method is very competitive in serial environments and offers the potential for vector-based parallel implementations. In Fast Decoupled Load Flow (FDLF) studies, a significant speed up with vector processing was reported compared to scalar processing [4, 5]. Thus, appropriate improvements in vector hardware, such as pipelining, pre-fetching, and vector chaining, have the potential to speed up significantly the above solution process. FPGAs that allow reconfiguration of hardware resources could be used in realizing vector machines suitable for the W-matrix method. Since the matrices used in power analysis are very sparse [2, 3], further opportunities are present for even better performance. Special techniques have been reported in the literature to take advantage of the sparsity in matrices [6, 4]. There is a clear demand for improved hardware platforms to enhance the performance of the W-matrix method. Current high-density FPGAs have the potential to satisfy this demand [7].

In this paper, our major objective becomes to incorporate special instructions to a vector processor in order to support the implementation of the W-matrix method. The proposed instructions and their FPGA implementation are presented, and actual performance results are analyzed. Matrix-vector multiplication is common in this problem. One reported FPGA implementation of sparse matrix-vector multiplication [8] lacks the generality of a programmable processor. Our approach is more general, involving a programmable vector processor to carry out many other tasks in addition to sparse matrix-vector multiplication. The effect on the performance of vector chaining is investigated. As there is inherent data parallelism in vector operations, the use of multiple functional units and multiple data buses is also studied. The effect of using a large vector register file is analyzed as well. The performance of our vector computer is measured for the 14-, 30-, 57-, 118- and 300-bus IEEE test systems.

2. The W-matrix Method and Required Vector Processing Support

Let A = L.D.U, $L^{-1} = W^L$ and $U^1 = W^U$. Then, the solution to the above problem can be obtained as $x = W^U.D^{-1}.W^L.b$ [4, 2, 3]. The solution is carried out in a series of matrix-vector multiplications: $W^L.b = z$, $D^{-1}.z = y$, and $W^U.y = x$. Each W-matrix can be partitioned to increase sparsity [2]. Appropriate node ordering has been used to minimize the number of non-zero elements in the W matrix [2, 3]. An efficient way to *count the number of non-zero* elements in a matrix would speed up the ordering phase. It is also imperative to support *sorting by magnitude* in order to speed up the ordering phase. To process only the non-zero elements, the capability of the architecture to *identify the non-zero elements* and *form a vector* would be meaningful. The capability to *form a vector with the column indices* of non-zero elements is also an important requirement. The W-matrix method involves a series of sparse matrix - vector multiplications. As the vector is formed exclusively from non-zero

elements belonging to different rows, the resulting vector also contains elements for different rows of the result. Multiplication result elements belonging to the same row must be added together to find the appropriate row element in the result. A design is needed to selectively *add elements of a vector.* This also requires keeping a record of non-zero elements on each row of the original matrix. In many applications, the contribution of diagonal elements is evaluated separately from the contribution of off-diagonal elements. This can be carried out efficiently if the diagonal elements are readily accessible to the processor. So, a technique is needed *to access only the diagonal elements.* Based on these arguments, we have decided to add the instructions summarized in Table 1 to the basic general-purpose vector architecture proposed in [9]. They offer potential performance improvement when implemented directly in hardware.

In our case, the vector computer consists of a vector processor with pipelined 32-bit FPUs, a memory controller for pre-fetching, and several memory modules, as shown in Fig. 1. It contains a program memory and eight data memory modules. There are six vector registers each having 16 elements of 32 bits. The vector architecture with the instructions in Table 1 was modeled in VHDL. Modelsim from Mentor Graphics was used to represent the model and simulate the design. Subsequently it was synthesized using Synplify-Pro from Synplicity. Finally, it was mapped to a Xilinx Virtex II FPGA using the Xilinx 'Place and Route' tool.

Table 1. Proposed vector operations to speed up the W-matrix method

Operation	Effect
Count the number of non-zero elements in a matrix	Fast ordering of rows for more parallelism and effective vector length control
Sort the rows based on the above counts	
Select a row based on the minimum count	
Add only certain elements of a vector to certain other elements of the ***same vector***	Fast multiplication of a sparse matrix with a vector
Access only the diagonal elements	Fast access of the elements
Create a vector from all non-zero elements in a matrix	
Create a vector from the column indices of all non-zero elements in a matrix	
Create a vector with the number of non-zero elements on each row	

3. Test Strategy and Performance Evaluation

The test cases chosen for performance evaluation are IEEE power systems. Their admittance-matrix characteristics are summarized in Table 2. In this first

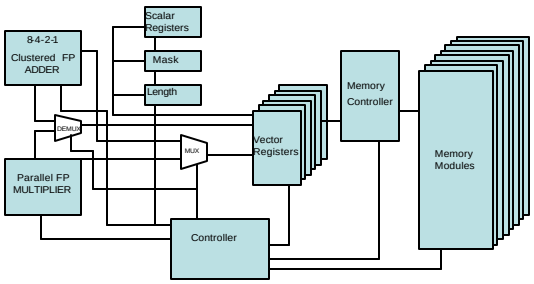

Figure 1. The vector accelerator

experiment, as a precursor to power flow calculations it is required to calculate the bus currents using a matrix-vector multiplication: $I_{bus} = Y_{bus} * V_{bus}$, where Y_{bus} is an admittance matrix and V_{bus} is a voltage vector [10]. The host program loads the bus vector and the admittance matrix values into the data memory of the vector processor. It also loads a predetermined set of vector instructions into its code memory. The vector processor executes the code to produce the resulting current vector and stores it in the data memory. The host program sets the vector length (for load and store operations) depending on the part of the matrix being executed. Results are read back into the host for verification. The timing is measured in processor clock cycles.

Table 2: Characteristics of IEEE bus systems

Bus System	Non-zero Elements in Y_{bus}	Density (%)
14-bus	54	27.551
30-bus	112	12.444
57-bus	217	6.678
118-bus	490	3.519
300-bus	1122	1.246

3.1 Performance of Sparse Techniques

To evaluate the sparse techniques, all the admittance matrices in our experiments were first made sparse with density 2-7%. For each sparse matrix, a vector containing only the non-zero elements of the whole matrix is created. A column index vector is used to load the appropriate elements of the multiplier vector. Then, the multiplication is carried out and subsequently the partial results are added to form the elements of the final result. The cycles needed for bus current calculations were determined experimentally; actual results on the FPGA board are shown in Table 3. At the application level (for bus current calculations), it can be seen that the sparse handling techniques can reduce the overall cycles by 20-25%. Experiments were carried out to see the relationship between the matrix density and the cycles needed. The density was varied from 2 % to 7%, as shown in Figures 2-4. It can be seen from the figures that the execution

cycles vary almost linearly with the density of the matrix. For the 30- and 57-bus systems, partial results are added to form the final result. This overhead is significant for low-density matrices. As a result, the curve is non-linear at low densities. But for higher densities, the curve is linear as the above overhead becomes insignificant.

Table 3: Effect of sparse instructions

Test System	Cycles Needed		Cycle Savings (in %)
	With standard instructions	With sparsity handling instructions	
14-bus	2110	1563	25.924
30-bus	9834	7550	23.225
57-bus	36872	29218	20.758
118-bus	155152	121664	21.583
300-bus	941824	737938	21.647

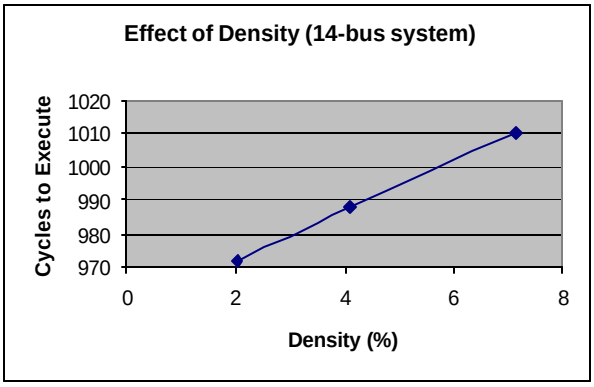

Figure 2: The effect of matrix density for the 14-bus system

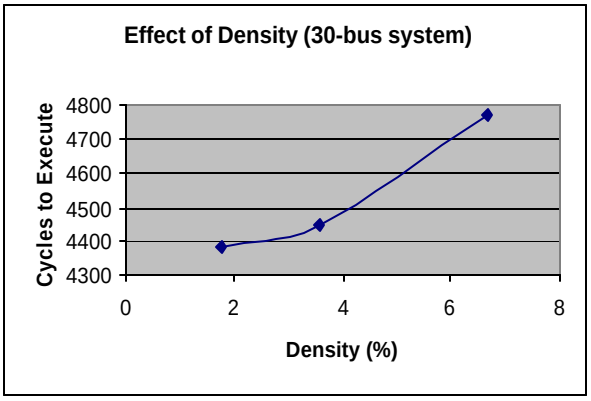

Figure 3: The effect of matrix density for the 30-bus system

3.2 Performance of Vector Chaining

The technique of forwarding the result of one vector operation to another vector operation is known as vector chaining. This technique offers a reduction in the cycles

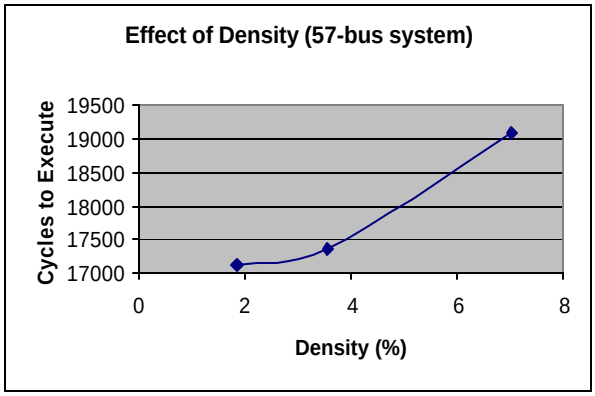

Figure 4: The effect of matrix density for the 57-bus system

required to fetch and decode an instruction chained to another instruction. Several vector chains with two and three stages were implemented and tested. Test results are summarized in Table 4. It can be seen that by employing vector chaining techniques we can save on overall cycles by 27-31%.

Table 4: Overall effect of vector-chaining

Test Systems	Cycles Needed		Cycle Savings (in %)
	Without the features	With the features	
14-bus	2110	1474	30.142
30-bus	9834	6734	31.523
57-bus	36872	26746	27.462
118-bus	155152	112540	27.464
300-bus	941824	681375	27.653

3.3 Performance of Multiple Functional Units and Data Buses

As all the elements of vector operand registers are simultaneously available, they are processed concurrently by employing multiple functional units in parallel in an 8-4-2-1 cluster. In these 4-stage clusters, each number represents the number of functional units employed at that stage. As 'Load' operations are more than multiplications, the data bus between the vector processor and the memory controller was broadened. As seen from Table 5, the effect of multiple functional units and data buses is favorable. However, the cycle savings fall for larger systems as there are more partial results that need to be loaded/stored and added.

Table 5: Combined effect of multiple FUs and DBs

Test System	Cycles Needed		Cycle Savings (in %)
	With Single FU, DB	With Multiple FUs, DBs	
14-bus	1010	812	19.603
30-bus	4771	3607	24.397
57-bus	19088	14094	26.163
118-bus	78010	58470	25.048
300-bus	458338	349330	23.783

The overall speed up obtained with our enhanced architecture is more than 2.5, as shown in Table 6.

Table 6: Overall Speed-up

Test System	Cycles Needed		Speed-up
	With Standard Architecture	With Enhanced Architecture	
14-bus	2110	812	2.598
30-bus	9834	3607	2.726
57-bus	36872	14094	2.616
118-bus	155152	58470	2.653
300-bus	941824	349330	2.696

4. Further Enhancements

Additional gain in performance could be achieved by using larger vector registers, such as 32-element registers. It reduces the load/store overheads and produces long vectors utilizing the pipeline more efficiently. However, it requires additional resources on the FPGA and currently efforts are ongoing to accommodate such a machine within one Virtex II FPGA. An estimate of cycle savings arising out of the use of the 32-element machine is presented in Table 7. As the admittance matrices are symmetric, loading a row implies loading a column too. It implies that the subsequent load lengths could be shorter if we could save the necessary elements from previous load operations. It could result in additional cycle savings. Also, load request (prefetch), count non-zero elements, and actual load operations are carried out sequentially for every row. Thus, chaining them together implies cycle savings for the whole process. Since the partitioned W_i is actually partitioned L_i (where i is the partition number) with the signs of the off-diagonal elements reversed, an instruction to produce W_i directly from L_i could be meaningful. These are in active consideration for implementation.

Table 7: Effect of Larger Vector Registers (estimated from code structure)

Test System	Cycles Needed		Cycle Savings (in %)
	On 16-element Registers	On 32-element Registers	
30-bus	3607	2645	26.670
57-bus	14094	10779	23.520
118-bus	58470	43430	25.722
300-bus	349330	272825	21.900

5. Conclusion

The objective was to provide hardware support to the W-matrix method found in power flow studies. An FPGA-based vector implementation for this method was presented. It was shown experimentally that about 22% of vector processor clock cycles could be saved by employing our proposed instructions. Also, the cycles needed to solve the current equations vary linearly with the matrix density. Moreover, by chaining several vector instructions it was possible to reduce the cycle requirements by about 28%. More than 20% cycles can be saved by employing multiple functional units with a wide data bus between the vector processor and the memory controller. An overall speed up of more than 2.5 was achieved in our experiments.

6. Acknowledgements

This research was supported in part by the U.S. Dept. of Energy under grant DE-FG02-03CH11171.

References

[1] D.J. Tylavsky and A. Bose, Parallel Processing in Power System Computation. *IEEE Trans. Power Sys.*, Vol. 7, No. 2, May 1992.

[2] M.K. Enns, W.F. Tinney, and F.L. Alvarado, Sparse Matrix Inverse Factors. *IEEE Trans. Power Sys.*, Vol. 5, No. 2, May 1990.

[3] F.L. Alvarado, D.C. Yu, and R. Betancourt, Partitioned Sparse A^{-1} Method. *IEEE Trans. Power Sys.*, Vol. 5, No. 2, May 1990.

[4] G.P. Granelli, M. Montagna, and G.L. Pasini, A W-Matrix Based Fast Decoupled Load Flow for Contingency Studies on Vector Computers. *IEEE Trans. Power Sys.*, Vol. 8, No. 3, August 1993.

[5] A. Gomez and R. Betancourt, Implementation of the Fast Decoupled Load Flow on a Vector Computer. *IEEE Trans. Power Sys.*, Vol. 5, No. 3, Aug. 1990.

[6] H.S. Huang and C.N. Lu, Efficient Storage Scheme and Algorithms for W-Matrix Vector Multiplication on Vector Computers. *IEEE Trans. Power Sys.*, Vol. 9, No. 2, May 1994.

[7] X. Wang and S.G. Ziavras, Parallel LU Factorization of Sparse Matrices on FPGA- Based Configurable Computing Engines. *Concur. Comput.*, March 2004, pp.319-343.

[8] H. ElGindy and Y.-L.Shue, On Sparse Matrix Vector Multiplication with FPGA Based System. *11th Ann.l IEEE Symp. Field Prog. Custom Comp. Mach*, April 2002.

[9] J.L. Hennessy and D.A. Patterson, Computer Architecture: A Quantitative Approach (Second Edition). *Morgan Kauffman Pub. Inc.*, 1996.

[10] J. Duncan Glover and M. Sarma, Power System Analysis and Design: with Personal Computer Applications. *PWS Publishers*, Boston, 1987.