Gauss-Seidel Accelerated: Implementing Flow Solvers on Field Programmable Gate Arrays

David P. Chassin, *Senior Member, IEEE*, Peter R. Armstrong,
Daniel G. Chavarría-Miranda, and Ross T. Guttromson, *Senior Member, IEEE*

***Abstract*–Non-linear steady-state power flow solvers have typically relied on the Newton-Raphson method to efficiently compute solutions on today's computer systems. Field Programmable Gate Array (FPGA) devices, which have recently been integrated into high-performance computers by major computer system vendors, offer an opportunity to significantly increase the performance of power flow solvers. However, only some algorithms are suitable for an FPGA implementation. The Gauss-Seidel method of solving the AC power flow problem is an excellent example of such an opportunity. In this paper we discuss algorithmic design considerations, optimization, implementation, and performance results of the implementation of the Gauss-Seidel method running on a Silicon Graphics Inc. Altix-350 computer equipped with a Xilinx Virtex II 6000 FPGA.**

***Index Terms*–Gauss-Seidel method, Field programmable gate arrays, Power flow**

I. Introduction

The power-flow problem is one of the most fundamental calculations in power systems engineering and operation. The computation determines the voltage magnitude and phase angle at each bus in power system under balanced three-phase steady-state conditions [1]. The information inputs for such a calculation include real and reactive power injections, real and reactive loads, line impedances, and network topology. After completing a power-flow calculation, important information about the power system can be derived, such as real and reactive power flows in equipment and line losses.

Iterative solutions to algebraic equations of the form $\mathbf{y} = \mathbf{f}(\mathbf{x})$ for power systems use an $N \times N$ invertible matrix $\mathbf{D}$ such that $\mathbf{x}_{t+1} = \mathbf{x}_t + \mathbf{D}^{-1}[\mathbf{y}–\mathbf{f}(\mathbf{x}_t)]$. In cases where $\mathbf{f}(\mathbf{x})$ is non-linear the preferred solution method is the Newton-Raphson (NR) algorithm, which iterates using at least the two lowest order terms of a Taylor series expansion of $\mathbf{f}(\mathbf{x})$ about an operating point $\mathbf{x}_0$, such that

$$\mathbf{y} = \mathbf{x}_0 + \left.\frac{d\mathbf{f}}{d\mathbf{x}}\right|_{\mathbf{x}=\mathbf{x}_0} \left(\mathbf{x} - \mathbf{x}_0\right).$$

Replacing the $\mathbf{x}_0$ by $\mathbf{x}_{t-1}$, we obtain the NR iteration equation

$$\mathbf{x}_t = \mathbf{x}_{t-1} + \mathbf{J}_{t-1}^{-1}\left[\mathbf{y} - \mathbf{f}\left(\mathbf{x}_{t-1}\right)\right]$$

where $\mathbf{J}_t^{-1}$ is the Jacobian of the system at the step t. The solution is said to have converged and iteration is stopped when $|x_{t-1} - x| \le \varepsilon$. If the solver fails to converge within a finite number of iterations the solver is stopped and a convergence error is flagged.

When the second order term is included the convergence performance of the NR method is quadratic. In cases where only the first-order linear term of the Taylor series expansion of $\mathbf{f}(\mathbf{x})$ is used, the NR method reduces to the Gauss-Seidel method and the convergence performance is linear, and usually requires more iterations to reach a given solution error ε.

The NR method is generally preferred in power flow calculations because of its quadratic convergence properties but it can have difficulties with a zero-angle, one per-unit initial condition—the so-called "flat" voltage start condition. On the other hand, the GS method convergence time increases significantly for large systems, and can exhibit convergence problems for system with high active power transfers [2]. Often the two algorithms' complementary strengths mean that power-flow programs implement both, using the GS method to rapidly determine an approximate solution from a flat voltage start, and then using the NR method to obtain the final well-converged and accurate solution.

However, from the standpoint of numeric processing, the calculation of the Jacobian requires access to data

This work was funded by U.S. Department of Energy's Pacific Northwest National Laboratory under the Energy Systems Transformation Initiative. Pacific Northwest National Laboratory is operated by Battelle Memorial Institute under Contract DE-ACO6-76RL01830.

David P. Chassin (david.chassin@pnl.gov), Peter R. Armstrong (peter.armstrong@pnl.gov), Daniel G. Chavarría-Miranda (daniel.chavarria@pnl.gov) and Ross T. Guttromson (ross.guttromson@pnl.gov), Pacific Northwest National Laboratory, Richland, Washington, 99352.

about every bus on each iteration, which is a serious obstacle to scalable implementations of the NR algorithm on many platforms, including FPGAs. The GS method does not suffer from this global data access problem, and is therefore simpler to implement on a gate array, in spite of convergence rate 1/10 or worse that of NR. Consequently, as long as a single iteration of GS is amply more than 10 times faster than an iteration of NR, we can expect a GS implementation on gate-arrays to outperform NR implementations. Given the right conditions, a parallel implementation of GS could allow much faster iterations times than NR. In particular, the topological characteristics of power grids lead us to expect the convergence speed to be roughly order ½ to the number of nodes, i.e., $t_{max} \propto N^{1/2}$.

The difficulty with such a high level of parallelization is twofold. First, the programming tools for reconfigurable hardware were designed to address digital signal processing problems, rather than this type of numerical problem. Second, complete parallelization of the computation requires very large gate-arrays (by the standard of today's technology), even for the most modest networks. However, neither of these two constraints is insurmountable within the foreseeable future, and thus we believe the results discussed here are timely. Demand for high-performance numeric processing tools should drive the market for the former, and Moore's law can be expected to remedy the latter in the course of time. We address these remaining difficulties by carefully considering how to design and optimize the GS algorithm on a gate-array.

II. Algorithm Design

While the GS method for solving systems of non-linear equations is generally well suited to implementation on an FPGA, it is not generally stated in a form that is ideally suited to reconfigurable hardware. We therefore consider what changes to the algorithm itself might improve speed, accuracy, and gate-array utilization.

To perform a single GS iteration, we perform the following operation at each bus *k* of the network

$$\overline{V}_k \leftarrow \frac{1}{\overline{Y}_{kk}}\left(\frac{\overline{S}_k}{\overline{V}_k^*} - \sum_{\substack{n=1\\n\neq k}}^{N} \overline{I}_{kn}\right) \tag{1}$$

where $\overline{V}_k$ is the complex voltage at the bus *k*, $\overline{Y}_{kn}$ is the admittance matrix term *kn*, and $\overline{S}_k = P_k + jQ_k$, with *P* the real power and *Q* the reactive power injections at the bus *k*. At each branch *kn* between bus *k* and bus *n*, we also perform the following operation

$$\overline{I}_{kn} = \overline{V}_n \overline{Y}_{kn}. \tag{2}$$

Expanding (2) and regrouping (1) we obtain

$$\overline{V}_k \leftarrow \left(\frac{S_k}{\overline{Y}_{kk}\overline{V}_k^*} - \sum_{\substack{n=1\\n\neq k}}^{N} \overline{V}_n \frac{\overline{Y}_{kn}}{\overline{Y}_{kk}}\right) \tag{3}$$

Using the following functions for scalar transformations:

$$\begin{aligned} \overline{G}(\overline{x}) &= \frac{\overline{S}}{\overline{Y}_{kk}\overline{x}^*} \\ \overline{H}(\overline{x}) &= \overline{x}\frac{\overline{Y}_{kn}}{\overline{Y}_{kk}} \end{aligned} \tag{4}$$

we can rewrite (3) as

$$\overline{V}_k \leftarrow \overline{G}(\overline{V}_k) - \sum_{\substack{n=1\\n\neq k}}^{N} \overline{H}(\overline{V}_n). \tag{5}$$

The $\overline{G}$ operation is performed once per iteration per bus, and the $\overline{H}$ operation is performed twice per iteration per branch (once in each direction). The inputs and outputs of $\overline{G}$, and $\overline{H}$ are rectangular complex values. However internally, the operation is more efficiently performed as polar calculation. Therefore conversions between polar and rectangular are frequently performed. This requires the implementation of divisions and arc-tangent calculations, which can be resource intensive on FPGAs.

III. Implementation

We have implemented a prototype Gauss-Seidel solver using the algorithm described above on an SGI RASC system at the Pacific Northwest National Laboratory (PNNL). This system is an standard Altix 350 with 8 1.5 GHz Itanium 2 processors, coupled together with a Virtex II 6000 FPGA board connected to SGI's high-speed NUMAlink communication fabric. The Virtex II board has 3 2MB SRAM modules directly attached to it, which are accessible to the algorithms running on the FPGA. The host processors communicate and transfer data to the FPGA through PIO commands.

The host processors communicate with the FPGA via a software interface that enables writes and reads to the FPGA memory modules, as well as to a set of input and output registers that are mapped to internal FPGA registers. This software interface also enables reconfiguration of the FPGA during an application's execution.

Implementing an application on this platform requires creating two components: a software component that will run on the host processor(s) and an FPGA component that will execute in hardware on the Virtex II. The software component controls the execution of the FPGA component and provides input and output capabilities to it. We have implemented a Gauss-Seidel solver that runs entirely within the FPGA, the software application just provides control and output capabilities,

A. Algorithms & Data Structures

Our FPGA implementation utilizes a fixed power grid topology, that is, the matrices that describe the power grid are compiled directly as coefficients into the FPGA's registers and memory banks. The main reason for doing this is to increase performance and decrease the complexity of manipulating sparse data structures in the FPGA. We have used Celoxica's Handel-C [5] programming language and environment to develop our FPGA implementation, together with the Handel-C hardware interfaces provided by SGI.

Our prototype implementation solves a 5-bus system with 1 voltage-controlled bus, 1 swing bus and 3 regular buses. The inputs to the application are the real and reactive power, the reactive power limits, the admittance matrix and the initial voltages for each bus. In the general case, the admittance matrix has a sparse structure with non-zeros corresponding to the grid topology. Our implementation uses a dense representation for the admittance matrix to decrease complexity and increase performance at the cost of some extra storage.

Our implementation uses a 32 bit, fixed-point representation for all its data. Handel-C provides a fixed-point library that provides data structures and arithmetic operators for handling fixed-point data. Each real number is represented using a signed 16 bit word for its integral part and a signed 16 bit word for its fractional part. Complex numbers are represented using two real numbers for their real and imaginary components.

B. Arithmetic Operators

We have built a library of arithmetic operators for complex numbers and in some cases for real numbers as well. Handel-C provides library-based basic arithmetic operators for fixed-point data: addition, subtraction, multiplication and division. However, the area usage and performance of the division operator did not satisfy our needs and for this reason, we built our own division operator using a successive approximation iterative scheme. The division operator first computes the reciprocal of the divisor (each iteration produces four correct bits of the reciprocal of the divisor) and in a subsequent step, it multiplies the dividend by the computed reciprocal to obtain the result. For the 16-bit fractional precision used, this scheme needs only four iterations to compute the reciprocal. The initial value of the reciprocal is obtained using the following technique: we look at the position of the most significant 1 in the binary representation of the fixed-point number, we then approximate the reciprocal by computing a number that has as its only digit a binary 1 in the complementary position of the original most significant 1:

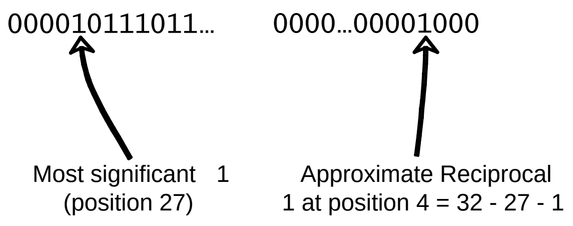

Fig. 4: Reciprocal Approximation

We also built a CORDIC library for fixed-point numbers to compute magnitudes and angles from rectangular representations of complex numbers, as well as to compute sines and cosines. This library is built on top of the basic fixed-point arithmetic operators, as well as some extra vendor-provided operations, such as: right and left shifts and relational operators (<, >, etc.). Based on the basic CORDIC computational template, we also implemented our own exponential operator, using a similar iterative and additive technique.

On top of all the previously described fixed-point operations we implemented all the required complex arithmetic operators: complex addition, subtraction, multiplication and division; computation of complex conjugates; computation of polar representations (magnitude and angle) from rectangular representations; and complex exponentiation.

C. Computational Structure

The implementation follows a similar iterative computational flow as would be used in a software implementation: first, the voltages for each non-swing bus are initialized from pre-compiled fixed-point ROM tables and constants; after this, the iterative solver begins; finally the results are written out to output registers on the FPGA that can be read by a host application.

In each iteration of the 5-bus prototype, the new voltages for the regular buses are computed in parallel with the new voltage for the single voltage-controlled bus in the system. This is feasible because we are using a modified form of the standard Gauss-Seidel formulation: all of the read voltage values in an iteration correspond to the values computed in the *previous* iteration, this enables greater parallelism and reduces the need for synchronization.

The computation of a single bus (regular or voltage-controlled) is implemented using several levels of internal parallelism:

- Two complex multiply/add units execute in parallel to compute the accumulated effect of the neighbors' contributions.
- For the voltage-controlled case, several rectangular-to-polar conversions execute in parallel, reducing their execution time at the expense of FPGA area.
- Many of the complex arithmetic operators have internal parallelism at an even lower level (i.e. several fixed-point products in parallel to implement the complex multiplication and division operators).

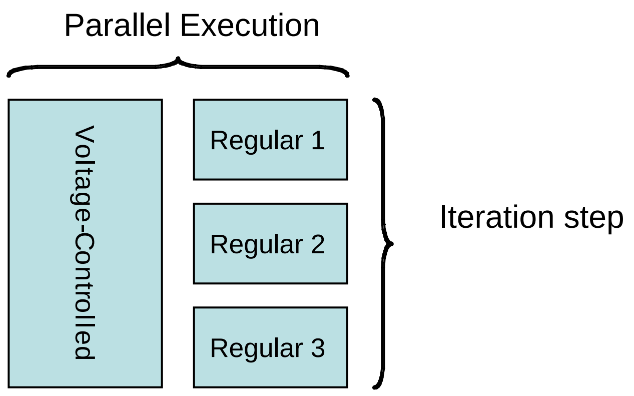

Fig. 5: Computational Structure

Since the computation of the voltage-controlled bus proceeds in parallel with the sequential computation of the remaining buses, several operators had to be replicated to enable parallelism and reduce synchronization: complex product, subtraction and addition operators, as well as the computation of complex conjugates. The complex division operator is shared between the two parallel control flow paths and a semaphore synchronizes access to it, due to Handel-C's restriction on simultaneous module usage. The two replicated complex product and addition operators are also protected by semaphores due to possible conflicts between the two control flow paths. These synchronization restrictions could be removed by replicating the operators at the cost of using more area on the processor, but we found that the current setup was a reasonable compromise between chip usage, speed and design synthesis feasibility.

D. Host Application

The software application driver executing on the Altix host processors has a very simple structure, since all input data is precompiled into the binary FPGA bitstream. It only has to load the bitstream image onto the FPGA from the SGI library manager, after that it starts the FPGA execution, waits for its completion and then reads the output registers to print out the results in fixed-point format.

IV. PERFORMANCE

Our implementation on the Virtex II 6000 FPGA executed at 100 MHz clock frequency, while using approximately 60% of the chip area. The implementation takes advantage of the built-in 18-bit hardware multipliers on the Virtex II to implement the fixed-point multiplication and division operators. We also use several of the on-chip block RAMs.

We have instrumented our implementation with a clock cycle counter that counts the cycles spent in the computational iteration section. The results are summarized in Table 1.

TABLE 1: PERFORMANCE RESULTS

Implementation	**Execution Time**	**FPGA Speedup**
Software Floating Point (1.5GHz)	3680μs	4.50
Software Fixed Point (1.5GHz)	2658μs	3.25
Hardware Fixed Point (100MHz)	818μs	1.00

The algorithm computes for 100 iterations for a total time of 818 microseconds (10^{-6} seconds). The computation of one iteration for a regular bus takes 2.13 microseconds; the computation for a voltage-controlled bus takes 8.09 microseconds per iteration. However, since the computations for the voltage-controlled bus and the three regular buses run in parallel the total execution time is closer to the execution time of the voltage-controlled bus only.

A software version of the same algorithm (written in C++) running on the 1.5 GHz Itanium 2 host processors of the Altix executes for 2658 microseconds, this version uses 32-bit fixed-point arithmetic similar to the hardware implementation. However, it utilizes the standard floating-point math library functions to compute sines, cosines and exponential functions. The software version was written using a template-based numeric class, which enables easy switching between fixed-point and floating-point implementations for debugging and testing purposes. The floating-point version executes for 3680 microseconds. The corresponding speedups are 3.25 and 4.5. Given the more than an order of magnitude difference in the clock frequencies of the FPGA with respect to the Itanium 2 processor, this is a significant acceleration factor.

Both hardware and software fixed-point versions are accurate to one fractional digit with respect to the floating-point version. The accuracy discrepancies are due to the more limited dynamic range of the 16:16 bit fixed-point representation compared to the range of the single-precision IEEE floating-point format.

V. Conclusion

We have demonstrated the feasibility of implementing a truly parallel non-linear network solver on reconfigurable computing hardware, such as FPGAs. The implementation presented also demonstrates a solution to the scaling problem when using intrinsically parallel implementation of algorithms on reconfigurable hardware. The time-to-solution of the current implementation of the Gauss-Seidel algorithm is expected to scale as the root of the network size until the capacity of the gate-array is reached, and after that it should scale as a traditional GS solver.

VI. References

[1] J. D. Glover, M. S. Sarma, *Power System Analysis and Design*, 3rd Edition, Brooks/Cole, Pacific Grove, California, 2002.

[2] P. Kundur, *Power System Stability and Control*, McGraw Hill, 1994.

[3] Eric W. Weisstein. "Square Root." From MathWorld—A Wolfram Web Resource. http://mathworld.wolfram.com/SquareRoot.html, 2005.

[4] R. Andraka, "A survey of CORDIC algorithms for FPGA based computers", in proc. of the 1998 ACM/SIGDA Sixth International Symposium on Field Programmable Gate Arrays, Monterey, California, 1998.

[5] Celoxica Corporation, "Handel-C Programming Language and Environment", http://www.celoxica.com

VII. Biographies

David P. Chassin (M '03, SM '05) received his BS of Building Science from Rensselaer Polytechnic Instituite in Troy, New York. He is staff scientist with the Energy Science & Technology Division at Pacific Northwest National Laboratory were he has worked since 1992. He was Vice-President of Development for Image Systems Technology from 1987 to 1992, where he pioneered a hybrid raster/vector computer aided design (CAD) technology called CAD Overlay™. He has experience in the development of building energy simulation and diagnostic systems, leading the development of Softdesk Energy and DOE's Whole Building Diagnostician. He has served on the International Alliance for Interoperability's Technical Advisory Group, chaired the Codes and Standards Group. His recent research focuses on emerging theories of complexity as they relate to high-performance simulation and modeling in building controls and power systems.

Peter R. Armstrong received the Ph.D. degree from Massachusetts Institute of Technology, Cambridge, in 2004. He is a Senior Research Engineer at Pacific Northwest National Laboratory where he has recently worked on efficient solvers for nonlinear network problems in the fluid, thermal, and electrical domains and on advances in acoustic anemometry. Other interests include fault detection involving transient models and nonlinear observers, adaptive models for optimal control in buildings and thermofluid processes, and analysis methods for estimating device or system parameters in the lab and in the field.

Daniel G. Chavarría-Miranda received the Ph.D. degree from Rice University, Houston, TX in January 2004. He is a Senior Research Scientist in the Applied Computer Science group at the Pacific Northwest National Laboratory where he currently is working on applying high-performance hybrid architectures to scientific computing. He is also working on programming language and run-time models for high-performance computing.

Ross T. Guttromson (M'01 SM'04) received his B.S.E.E. and M.S.E.E. degrees from Washington State University. Currently, he is a Senior Research Engineer with the Energy Science and Technology Directorate at the Pacific Northwest National Laboratory, Richland, WA. He was previously with R.W. Beck Engineering and Consulting, Seattle, WA, from 1999 to 2001 and with the Generator Engineering Design Group, Siemens-Westinghouse Power Corporation, Orlando, FL, from 1995 to 1999. Mr. Guttromson previously served on the nuclear submarine USS Tautog (SSN 639), and is a licensed Professional Engineer in the state of Washington.