Improving Analog Simulation Speed using Selective Matrix Update

Sameer S. Kher, Harold W. Carter
Distributed Processing Laboratory
ECECS, University of Cincinnati
Cincinnati, OH – 45221-0030, USA
{kherss, hcarter}@ececs.uc.edu

Abstract – Dynamic systems of nonlinear differential algebraic equations (DAE) are used to model a large class of engineering and biological systems. The two most time consuming processes in the direct method for analysis of nonlinear DAE sets are found to be the Jacobian matrix formulation and the linear system solution. Methods are presented in this paper to significantly reduce Jacobian matrix formulation times, and hence improve simulation speeds, by selective update of the Jacobian matrix.

Introduction

Most physical systems exhibit varying behavior depending on current values of various parameters in the system. Modeling such discontinuous dynamic behavior requires different sets of DAEs, where the behavior of the system is determined by the current DAE set, which in turn is determined by the current variable values. As an example, a transistor is commonly modeled with three behavioral regions where the DAE sets differ for each region (1).

Contemporary simulators rely on a hierarchy of equation sets. At each simulation cycle, a path through this hierarchy is traversed to form the current DAE set. Simulators perform an incremental analysis over time on these dynamic DAE sets by dividing the simulation time into time steps and solving the DAE sets at each time step (2).

The commonly-used direct method for analysis of the nonlinear DAEs performs repeated iterations each involving the following steps. First, the equations are integrated using implicit integration methods to convert the nonlinear DAE set into a set of nonlinear algebraic difference equations (AE). The AE's are then linearized, typically using an iterative Newton-Raphson (NR) method, The NR method involves calculating the Jacobian Matrix and the right hand side (RHS) vector, to form a linear system, represented as **Ax**=**b**, where **A** represents the Jacobian matrix, **x** is a vector of the unknowns and **b** is the RHS vector. Finally the equation **Ax=b** can be solved by some form of Gaussian Elimination.

Research has shown the two most time consuming processes in the direct method to be the Jacobian matrix formulation and the linear system solution (3). Simulators that rely on built-in mathematical models of a fixed set of circuit components, such as SPICE, have relatively fast Jacobian matrix build times since the Jacobian does not need to be explicitly calculated. However, higher level modeling languages allow the modeler to specify arbitrary differential and algebraic equations. Formulation of the Jacobian matrix for such dynamic arbitrary systems requires some sort of symbolic or automatic differentiation (4) (5). In addition, typically, due to the dynamic nature of the system, the Jacobian matrix and the RHS vector have to be rebuilt for every iteration of the NR method. Thus, the Jacobian matrix build times for simulators for these languages are found to contribute a significant portion of the total simulation time. Hence, simulation speeds can be significantly improved by reducing matrix formulation time.

Previous work includes (3), where matrix build and solve times were compared for an analog circuit simulator with built-in models and (6) and (7), which demonstrated Jacobian matrix build speed improvements where the entire DAE set can be classified. We extend this effort to include individual equation types for which the number of calls to the automatic differentiator can be reduced and to handle dynamic systems by mapping the hierarchical path to the best possible solution. Similar concepts are addressed in (8) but we more specifically exploit linearity and time-variance to support reduced build times.

The methods were implemented for SIERRA-2.0[1] and resulting improvements in simulation times are reported here. We report results for two types of scalable models; the first type consists of models with 1000 components and varying percentages of linear/nonlinear and time-variant/time-invariant equations and the second type is the input stage of a multi-channel data acquisition system.

System Overview

Fig. 1 shows the simulation method traditionally used for continuous-time simulation in case of a mixed-mode system. The input to the simulator is the elaborated set of equations as well as any discontinuity conditions. The DC operating point is calculated to obtain initial values for all dependent variables.

[1] VHDL-AMS simulator under development at the University of Cincinnati

The transient simulation is then started with the current time set to 0. Initialization discontinuities are processed and values are set accordingly. The current set of DAE's is then determined by tracing the appropriate path through the elaborated hierarchical data structure of equations. In addition to the set of equations specified by the modeler, systems often also obey some conservative structural laws. For example, electrical systems obey Kirchoff's current law (KCL). These laws give rise to an additional set of equations, which are required to completely describe the behavior of the system and are also loaded into the Jacobian matrix.

Next, the differential equations are integrated to give a set of nonlinear AEs. These equations are then solved using the iterative NR method, which involves the building of the Jacobian Matrix followed by linear system solve, during each iteration at every time step.

The NR method iterates until convergence criteria, determined by desired tolerances, are met. The time step control then determines the next time step for simulation and the new equation set is obtained and solved. Simulation continues until the specified end time.

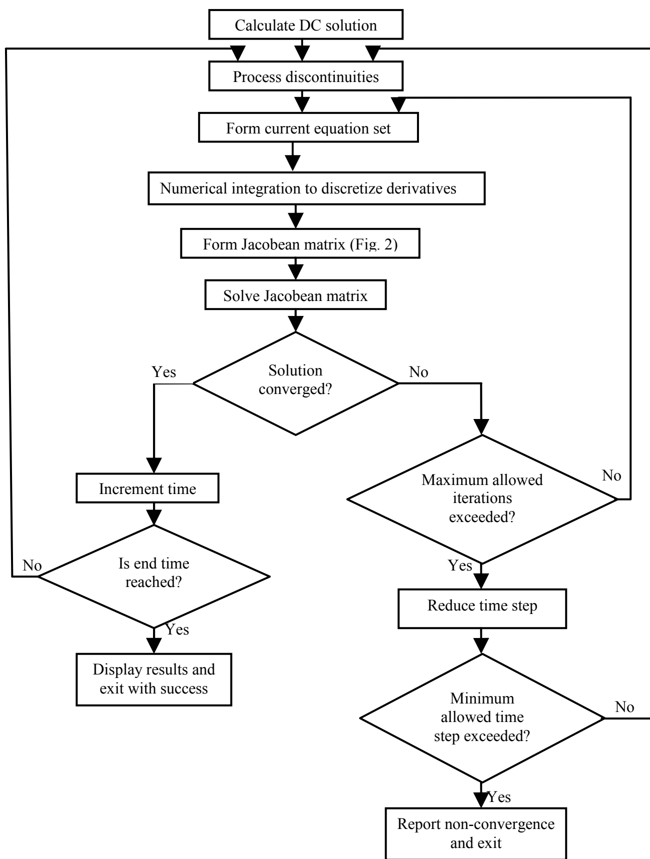

Figure 1: Traditional continuous simulation method. Jacobian matrix formulation procedure is shown in Figure 2 (and Figure 4 in improved form)

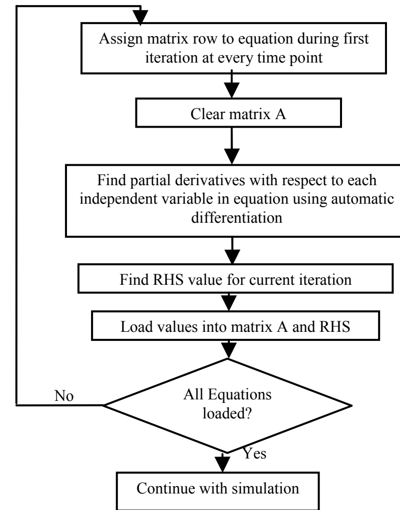

Figure 2: Traditional Jacobian matrix formulation.

Fig. 2 shows the traditional Jacobian Matrix formulation method. The current set of equations is the input to this method. First, matrix locations are allocated to each term in each equation. The Jacobian matrix is then cleared at the start of each iteration. Finally, the AE's are loaded one by one by determining the partial derivatives of each equation with respect to each of its variables by using an automatic differentiator. This process is slow since it is traditionally performed at each time step.

Approach

One method of improving Jacobian matrix build times is by classifying the entire equation set at every time point as either linear or nonlinear and applying appropriate solution methods.

However, most physical systems exhibit some nonlinear behavior and classification of the entire system as linear is rare. Our modification to the traditional algorithm reduces the time required to build the Jacobian matrix for dynamic nonlinear systems by exploiting the properties of individual equations.

For a nonlinear DAE set, we identify matrix locations that can remain unchanged by classifying the equations. A number of attributes are associated with every equation such as the number of variables in it, whether it is a differential equation or an algebraic equation, etc. In order to determine the matrix locations that can remain unchanged between time-steps, we consider two of the equations' attributes namely the linearity and the time variance. These attributes determine the classification of the DAEs.

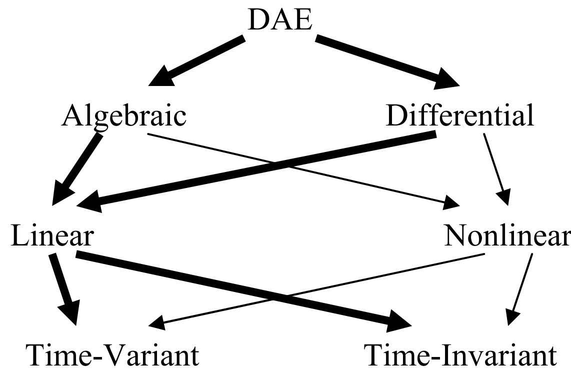

Figure 3: Equation type hierarchy. Thick lines represent the two equation type combinations explored thus far and reported here.

Classification of Equations:

DAEs can be classified as shown in Fig. 3. Algebraic and differential linear equations are classified as linear time-variant or linear time-invariant depending on the presence of time as a variable in the equation. All other equations are classified as nonlinear.

Selective matrix update during simulation

Simulation occurs in two phases: model compilation and model simulation. The equations specified by the modeler are classified during compilation. During simulation, while selecting the equations to be included at the current time point, we determine if the equation set has changed from the previous iteration. We may also classify the system as linear or nonlinear depending on the equations to be solved. The classification may then be used to select one of many solution methods as suggested by (6).

If the system is nonlinear, we use the classification of individual equations to reduce the build time. Fig. 4 shows the modified Jacobian matrix formulation method. In addition to the current equation set, we also indicate if the equation set has changed from the previous iteration.

Allocation of matrix entries to equations is done during the first iteration and then only when the equation set has changed in subsequent iterations.

If an equation is linear and time-invariant, we calculate the coefficients and RHS only for the first iteration at the first time point. These values are then stored in the matrix and need not be changed until the equation set changes.

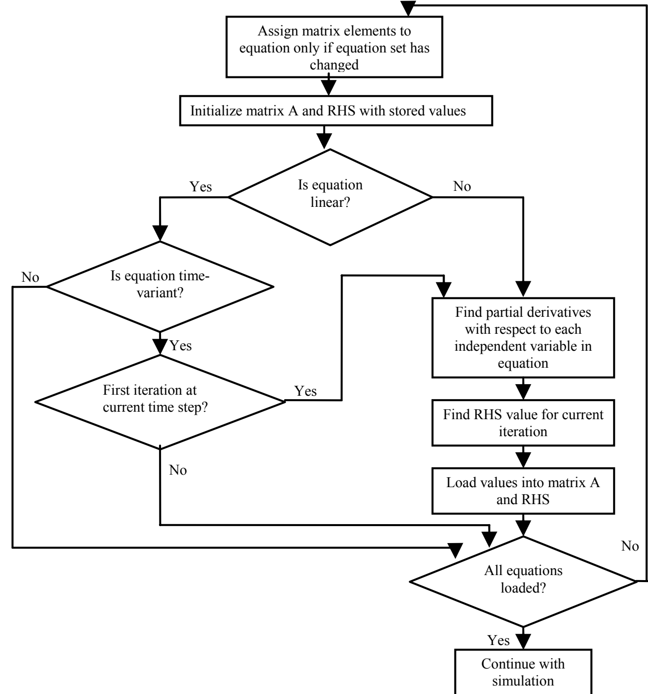

Figure 4: Improved Jacobian matrix formulation

If an equation is linear and time-variant, we calculate the Jacobian and RHS only for the first NR iteration at each time point. These values are then stored in the matrix and are reused for the remaining iterations until convergence at the current time point.

Nonlinear equations are loaded using the traditional method.

Simulation Results

The proposed algorithm was implemented on the VHDL-AMS Simulator SIERRA-2.0, available from the Distributed Processing Laboratory, University of Cincinnati.

The first set of models was obtained from a 1000-component model generator (9). The models were generated with the desired percentages of linear time-invariant, linear time-variant and nonlinear equations for analysis. Components

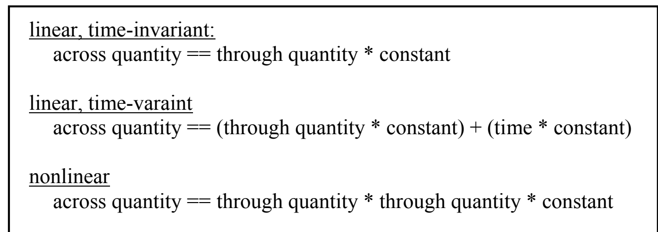

Figure 5: Component model equations used for linear and nonlinear components in 1000 component model

were modeled as simple simultaneous statements as shown in Fig. 5.

Table 1 shows the simulation times and the percentage speedup obtained over the traditional method for simulation on a 1.0 GHz AMD-Athlon processor with 128MB memory executing Red Hat Linux 7.3.

Table 1: Simulation results for 1000-component model

% Linear Equations (out of 1000)*	% Time-variant Equations (out of linear equations)**	Selective Matrix Update		Traditional Method		% Speedup in Total Simulation Time
		Matrix Build Time (seconds)	Total Simulation Time (seconds)	Matrix Build Time (seconds)	Total Simulation Time (seconds)	
0	0	53.46	83.3	53.46	83.3	0.00
25	0	43.87	74.51	52.38	83.01	10.24
	50	46.4	78.88	53.44	85.9	8.17
	100	47.3	79.01	52.23	84.02	5.96
50	0	34.76	68.36	53.76	86.91	21.34
	50	38.92	70.95	55.47	87.54	18.95
	100	41.27	74.77	54.22	87.01	14.07
100	0	7.17	43.1	53.6	86.91	50.41
	50	13.19	48.4	54.28	87.13	44.45
	100	20.49	53.74	53.34	85.5	37.15

* remaining equations are modeled as nonlinear
** remaining linear equations are linear time-invariant

In each model the remaining percentage of equations were nonlinear. As expected, a completely linear time-invariant system gives the highest speedup.

Fig. 6 shows effect of increasing the number of linear time-invariant equations on speedup and Fig. 7 shows effect of increasing the number of linear time-variant equations on speedup. Both graphs show increasing speedups for increasing number of classifiable equations. We also note that the rate of improvement due to linear time-invariant equations is greater than that for linear time-variant.

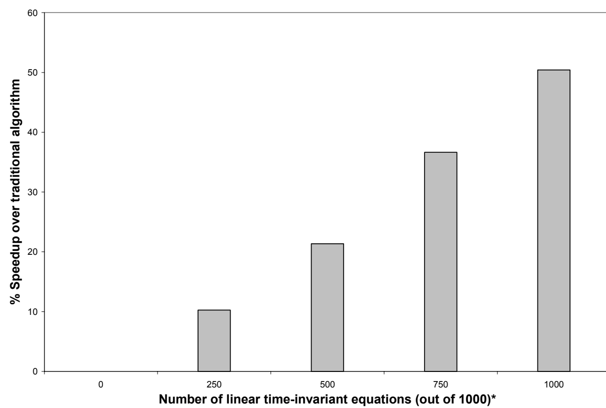

Figure 6: % Speedup in total simulation time with respect to number of linear time-invariant equations for 1000 component model.
***remaining equations are nonlinear.**

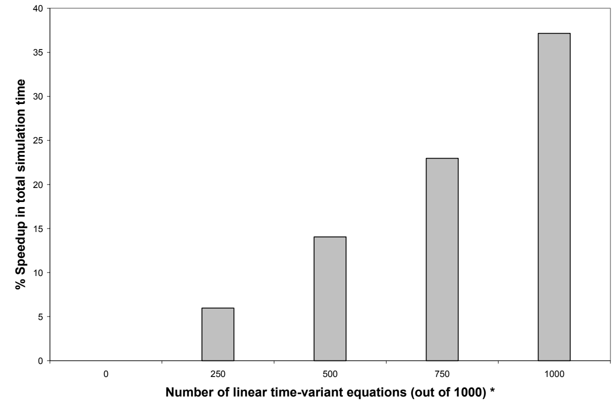

Figure 7: % Speedup in total simulation time with respect to number of linear time-variant equations for 1000 component model.
***remaining equations are nonlinear.**

Increasing the percentage of linear time-variant equations with respect to linear equations decreases the speedup. This is because the automatic differentiator is called during the first NR iteration at every time step instead of only once at the start of simulation.

Fig. 8 shows the effect of increasing the number of linear time-variant equations with respect to the linear time-invariant equations on speedup.

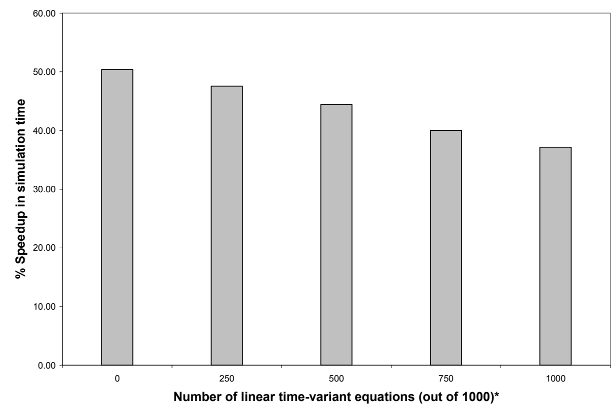

Figure 8: % Speedup in total simulation time with

The second set of models represents the input stage of a multi-channel data acquisition system. Each channel accepts a time varying sinusoidal signal as input. The input signal is rectified by a half-wave rectifier and is used to drive a load resistance. The half-wave rectifier uses a single diode, which is modeled to exhibit three operational regions - a nonlinear forward bias region, a linear reverse bias region and a

nonlinear breakdown region. Thus, this stage has three classifiable equations, a linear time-variant input signal, a linear time-invariant reverse bias region and a linear time-invariant load resistance equation.

Table 2 shows the simulation times and the percentage speedup obtained over the traditional method for simulation on a 1.0 GHz AMD-Athlon processor with 128MB memory executing Red Hat Linux 7.3.

Table 2: Simulation results for multi-channel input stage for data acquisition system

Number of Input stages	Selective Matrix Update		Traditional Method		% Speedup in Total Simulation Time
	Matrix Build Time (seconds)	Total Simulation Time (seconds)	Matrix Build Time (seconds)	Total Simulation Time (seconds)	
1	0.67	0.75	1.29	1.37	45.50
2	1.15	1.26	2.40	2.51	49.84
4	2.19	2.33	4.64	4.79	51.46
8	4.24	4.49	9.34	9.59	53.18
16	8.85	9.34	19.39	19.88	53.03

Conclusions

We draw from the tables and graphs, that it is possible to significantly enhance the time and number of updates to the coefficient matrix and the RHS. In fact we have shown that speedups of up to 53% in total simulation time are possible over traditional methods for the benchmarks used. We also conclude that for systems with the same percentages of classifiable equations, greater speedups are achieved for systems requiring more number of iterations for solution convergence.

Future work will focus on techniques to increase the number of classifiable equations by using substitutions to convert nonlinear equations into one of the two classifiable forms.

References

(1) G. Massobrio and P. Antognetti, *Semiconductor device modeling with SPICE*, 2ed. McGraw-Hill, Inc., New York, NY, 1993.

(2) J. Vlach and K. Singhal, *Computer Methods for Circuit Analysis and Design,* second edition, Van Nostrand Reinhild, New York, NY, 1994.

(3) A. R. Newton and A. L Sangiovannni-Vincentelli, "Relaxation-based electrical simulation", *IEEE Transactions on Electron Devices,* Vol. ED-30, No. 9, September 1983.

(4) P. Frey, K. Nellayappan, V. Shanmugasundaram, R. S. Mayiladuthurai, C. L. Chandrashekar and H. W. Carter, "SEAMS: Simualtion environment fpr VHDL-AMS", *Proceedings of the 1998 Winter Simulation Conference,* Washington, D.C., United States.

(5) Griewank, A., D. Juedes, and J. Utke (1996, September), "ADOL-C: A Package for the Automatic Differentiation of Algorithms written in C/C++." in *ACM Transactions on Mathematical Software (TOMS)* Volume 22 , Issue 2 (June 1996).

(6) V. Shanmugasundaram, "*A dynamic multiple solution approach to improve the efficiency of VHDL-AMS simulation",* Master's Thesis, University of Cincinnati, 1998.

(7) S. Agarwal, "*Optimization approaches for analog kernel to speedup VHDL-AMS simulation*", Master's Thesis, University of Cincinnati, 2002.

(8) G. Hachtel, R. Brayton and F. Gustavson, "The sparse tableau approach to network analysis and design", *IEEE Transactions on circuit theory,* Vol. CT-18, No. 1, January 1971.

(9) S. Bapat, "The performance evaluation of VHDL-AMS simulators by creating large, scalable VHDL-AMS models", Master's Thesis, University of Cincinnati, 2002.