Proc. of The European Design Automation Conference (EDAC)
pp. 306-309; Glasgow, Scotland, march 12-15, 1990 (reformatted)

Multirate Integration in a Direct Simulation Method

J.T.J. van Eijndhoven   *M.T. van Stiphout*   *H.W. Buurman*

Eindhoven University of Technology
Department of Electrical Engineering
PO box 513, 5600 MB Eindhoven, The Netherlands
email: jos@es.ele.tue.nl

1. Abstract

Multirate integration is a technique in which a set of differential equations is solved with different timesteps assigned to subsets of equations [4][10]. In circuit simulation this is commonly used in the waveform relaxation method, where different subcircuits are analyzed independently from the others. An important and obvious advantage is the simulation efficiency: subcircuits which are temporarily changing relatively slowly, can be analyzed with large stepsizes, independent of the activity in other subcircuits. In this paper an approach is presented to fit multirate integration in a direct simulation scheme, thus bringing comparable advantages without the relaxation process and its related problems. An event driven scheme is proposed for the circuit simulation problem, with individual timesteps for every component in the circuit.

Only with the new combination of a highly efficient update scheme for the L/U decomposition, and some event clustering method, leads the multirate scheme to the expected speedup of the simulation process.

Keywords: Timing, Circuit Simulation, Multirate Integration.

2. The Idea

Before moving into a detailed analysis, a rough outline of the method is sketched. For circuit simulation, we emphasize the partitioning between one large sparse set of *network equations* and many small sets of *component equations*. The network equations are based on the Kirchhoff voltage and current laws, specifying linear relations between node voltages, and possibly branch currents or branch voltages. The component equations are in general small sets of nonlinear relations between the component terminal variables (node voltages and terminal currents) and internal state variables, causing a dynamic behavior.

By applying an integration formula and a linearization to these small sets of equations, the *jacobian* is determined for every component: A small set of linear equations which specifies the relation between the terminal variables of the component, and is assumed to be valid within some error bound during a small timeframe. All these jacobians are inserted in the set of (linear) network equations, which is subsequently solved for all the network (terminal) variables. The size of the time step h is adjusted during the analysis, controlled by an error estimation. So far, everything is standard practice. In our approach now, the size of the timesteps is assigned independently and individually for each component, thus yielding jacobians based on different values of h on every single timepoint. Every jacobian is assumed to be a valid representation of the component, for a time interval corresponding to its own h, starting at the jacobian creation time. The endpoint of this interval is set as *event* for this component.

To obtain a fast analysis scheme, we want to minimize the amount of work spent on slow parts of the circuit, i.e. parts with large values of h. The jacobian equations are built to *solve time derivatives* of the circuit variables. This allows to keep these *equations constant* for slowly changing subcircuits. Actually a first order interpolation scheme is obtained this way, creating only second order integration errors, which are constantly monitored and controlled. Solving for the actual circuit variables would give considerably less opportunities to keep parts of the equations constant, since it would induce first order errors which quickly become unexceptably large [9]. In our approach furthermore only modifications to these circuit variable derivatives are solved, with an efficient sparse implementation. This makes the amount of work for a single event proportional to its effect, instead of proportional to the size of the entire circuit. Now the number of operations (the CPU time) per event depends linearly on the actual number of modified circuit variables. Basically this constitutes a new and natural latency exploitation method.

The simulation algorithm now repeatedly determines the nearest event with the related *current component*, and increments the simulation time to this event, updating the values of the circuit variables. This is easily done, because the system keeps track of values for both each variable and its derivative. A new jacobian is now created for the current component, possibly with a modified value for its h, and a new event is set for it. The linear network equations are solved at the current timepoint, resulting in updated derivatives of the system variables.

Of course the components in the circuit influence each others behavior. To control these effects, the local truncation error of the applied integration formula is evaluated for all components whose terminal variable derivatives actually modify due to the event of this current component. If the truncation error grows, the event of the affected component is modified to an earlier time point, thus limiting the integration error to an upper bound.

3. Comparison with Others

As result a method is obtained which performs a numerically stable integration, and exploits latency in a natural way. The multirate integration scheme, which is the most important contributor to the speed of waveform relaxation programs, brings comparable advantages to this simulation method. Since we really solve sets of (implicit) network equations it is still a *direct* method, as opposed to waveform relaxation methods which rely on a converging relaxation process to solve the circuit variables. Due to the direct solution of equations we don't have to enforce strict circuit properties ('MOS with grounded capacitors') to obtain convergence, but can instead allow a wide freedom in component and circuit modelling such as analog and digital macromodels.
Due to the carefully implemented sparse *update* scheme the CPU time spent in the solution of the network equations stays relatively small.

Together with conventional techniques like 'spice' and the waveform methods we strictly adhere to stiffly stable implicit integration techniques. Many other approaches speed up simulation by performing a node-by-node integration technique, not solving network equations and discarding basic numerical stability and accuracy aspects. For reliability reasons we did choose that track.

In the next sections, the principles of our method will be treated in more detail, and results are presented.

4. Validity of the Jacobians

To explain the integration method, we will restrict ourselves here to linear systems. Every component of the electronic circuit can now be modeled with a matrix:

$$\begin{pmatrix} 0 \\ \dot{u} \end{pmatrix} = \begin{pmatrix} A & B \\ C & D \end{pmatrix} \cdot \begin{pmatrix} x \\ u \end{pmatrix} + \begin{pmatrix} a \\ b \end{pmatrix} \tag{1}$$

Here the vector x represents the terminal variables of the component, such as node voltages and branch currents. The matrices A B C D and the source vectors a and b constitute the model equations. The vector u is responsible for the dynamic behavior of the component, with $\dot{\mathbf{u}} = \partial u / \partial \mathbf{t}$. The dimension of the vector u will normally be small, typically 1 to 3. To this system an integration formula is added, for instance the trapezoidal rule:

$$u(t+h) = u(t) + \tfrac{1}{2} h \cdot (\dot{u}(t) + \dot{u}(t+h)) \tag{2}$$

For the multirate integration scheme, we introduce $\bar{u}$ and $\bar{x}$ as:

$$\bar{u} = \tfrac{1}{2}(\dot{u}(t) + \dot{u}(t+h)) = \frac{u(t+h) - u(t)}{h} \tag{3a}$$

$$\bar{x} = \frac{x(t+h) - x(t)}{h} \tag{3b}$$

If we want solve for $\bar{x}$ in our network equations, the following jacobian (4) is derived from (1)-(3):

$$J \cdot \bar{x} = j \tag{4a}$$

with:

$$J = A + \tfrac{1}{2} h \cdot B \cdot (I - \tfrac{1}{2} hD)^{-1} \cdot C \tag{4b}$$

$$j = -B \cdot (I - \tfrac{1}{2} hD)^{-1} \cdot \dot{u}(t) \tag{4c}$$

and I the identity matrix.

These jacobians are derived for all components -each with its own h- and substituted into the linear network equations. A sparse L/U decomposition is maintained from which the vector $\bar{x}$ is now solved by F/B substitution. An estimation of the integration error is now easily obtained: Differentiation of the component equations yields

$$\ddot{u} = C\dot{x} + D\dot{u} \tag{5}$$

$\bar{x}$ serves as relyable estimate for $\dot{\mathbf{x}}$, and $\dot{\mathbf{u}}$ is known from (1). The vector $\ddot{\mathbf{u}}$ is now used for the local truncation error:

$$LTE = K \cdot h^2 \, || \, \ddot{u} \, || \tag{6}$$

h is chosen in every component individually, to keep this LTE below some upper bound. When the corresponding jacobian is established, the event of this component is set at the current timepoint plus h. If the $\bar{x}$ changes during this time interval, the norm of $\ddot{\mathbf{u}}$ is recomputed. This might lead to the rescheduling of the event.

5. Event Processing

In this section we will discuss the interaction between different components, due to the network equations. To distinguish between components, a superscript k is introduced, whereas symbols without superscript refer from now on to the global network items. Thus we can write the network equations as:

$$S \cdot \bar{x} = s \tag{7}$$

S is a large sparse square matrix of full rank. Depending on the generation of these equations, i.e. sparse tableau analysis or modified nodal analysis, the matrix $(\mathbf{S}, s)$ contains somehow all the jacobians $(\mathbf{J}, j)^{\mathbf{k}}$. The terminal variables of the k'th component $\bar{x}^k$, are a subset of the vector $\bar{x}$. Clearly these subsets might overlap.

In an initial DC analysis phase, the vector $x(\mathbf{t} = 0)$ is solved. For all elements of this vector, the *time of last update* is recorded as $t_i = 0$ for all elements x_i individually. For all components k a stepsize h^k has been chosen, and the vector of derivatives $\bar{x}$ is determined.

During the transient analysis, the program repeatedly selects and solves the nearest event, until the final integration time T is reached:

The component k corresponding to the nearest event is selected. For this component, the norm of $\ddot{u}$ is determined, to decide whether h^k is kept equal, is enlarged, or is reduced.

h^k is kept equal:
In this case, the jacobian J^k (4b) doesn't change, and the matrix S and its L/U decomposition remain valid. However the right hand vector j^k (4c) will normally change, and will lead to an update Δs to the vector s. From Δs and the existing L/U decomposition of S, we compute $\Delta\bar{x}$, the update to $\bar{x}$. Because Δs originates from the jacobian of a single component, it will have very few (typically less than 3) nonzero elements. Therefore $\Delta\bar{x}$ will normally have very few nonzero elements, and a careful sparse implementation will yield this vector in very few operations, in principle independent on circuit size. Now for all indices i of nonzero elements in $\Delta\bar{x}$ an updating is done as:

$$\begin{aligned} x_i &= x_i + (t^{event} - t_i)\,\bar{x}_i \\ \bar{x}_i &= \bar{x}_i + \Delta\bar{x}_i \\ t_i &= t^{event} \\ &\text{print}(i, t_i, x_i) \end{aligned} \qquad (8)$$

For all components which are incident to these nonzero elements of $\Delta\bar{x}$, the integration errors are checked, and if required their events are reset according to (5) and (6).

h^k is modified:
It is either decreased due to a large $\ddot{u}$ requiring a small h^k for sufficient integration accuracy, or it is enlarged due to a small $\ddot{u}$ allowing a faster analysis. The new h^k leads to a different J^k according to (4b), and a different S, requiring a new L/U decomposition. The modification ΔS has a rank of at most the dimension of the vector u^k, typically 1 to 3, and a very small number of nonzero elements, typically less than 10. Therefore the new L/U decomposition is efficiently obtained with Bennetts algorithm [1], with an amount of work proportional to the number of modified elements in the L/U decomposition [2].
Of course the source vector j will normally also change. Therefore the modification $\Delta\bar{x}$ is a result from both the modification in S and in s. However $\Delta\bar{x}$ is still easily and efficiently found, in a cpu time which doesn't directly depend on the size of the circuit [2]. The other operations needed to solve this event correspond to the above case with unmodified h^k.

6. Event Synchronization

In the above algorithm, at every event the integration accuracy is checked in the components incident to the modifications. This checking is required but forms a considerable overhead in cpu time. To reduce this overhead, several events can be clustered on the same time point, and solved together. This would involve for a set of events a single solution of the sparse system, and a single scan through the incident components. Cpu time is saved, if components would have been reached by more than one event in the cluster, which is often true.

This clustering of events can be realized by a very simple scheme: Limit the choice of values for h^k to values of the form:

$$h^k = T \cdot 2^{-m}\ , \quad m \text{ positive integer} \qquad (9)$$

And limit the timepoints for the setting of new events to be an integer multiple of h^k. This way, components with comparable speed of change will automatically be assigned equal event points, and thus a strong clustering is obtained.

The event clustering together with the sparse matrix implementation can be viewed as a dynamic runtime partitioning method. In the limit situation of a small strongly coupled all-active network, the analysis scheme will automatically start to behave spice-like, because all components are evaluated at each timepoint.

7. Nonlinear Systems

For the analysis of electronic networks, the method must be extended to cope with nonlinear model equations. Basically this leads to two additional problems:
a) At every event the jacobian of the current component might change due to a new linearization.
b) Strong nonlinearities can lead to unacceptable truncation errors (LTE's) or even nonconvergence, requiring the reduction of the timestep h^k.

The renewed linearization can be accounted for in the L/U decomposition of S by an efficient sparse update. This process seems feasible for Newton Raphson based methods [6]. However our program is based on continuous piecewise linear component models, which easily allows for a rank one update every time the solution moves to a neighbouring segment of the model.

The reduction of h^k due to nonlinearities is an essential problem in a *fastest-first* multirate integration scheme as our implementation. Recognizing that h^k is too large at the time of the corresponding event leads to implementation problems, because other components with smaller values of h have already been integrated beyond the time point to which h_k should be reduced.
The alternative *slowest-first* method makes use of an extrapolation of the fast variables over a time interval which can be orders of magnitude larger than these fast time constants. This might lead to uncontrollable errors [4][10].
To solve this dilemma we employ the fastest-first method, requiring only extrapolations of signals within a time interval smaller than their time constant. To prevent the problem of having to backup in time for the fast signals, we check the nonlinearities in the components together with the LTE every time they are touched by an event of another component (see eqs. (6) and (8) and the subsequent paragraphs). This

rescheduling of events can however change the order of the integration method, if done frequently. This motivates the choice for a conservative second order formula for the LTE, even if the trapezoidal rule is used. Use of piecewise linear component models allows for an easy check for nonlinearity: With given $\bar{x}$ the time distance to the nearest boundary of the current linear segment can be explicitly determined, and the event can be set accordingly.

8. Results

Many basic aspects of the integration scheme explained in this paper are implemented and running in a circuit simulation program since 1984 [3][11], in which piecewise linear relations are used for all the nonlinear model equations. As integration formula, a choice is available for the trapezoidal rule, backward Euler, and BDF methods. The integration scheme behaves nicely, as far as can be expected from those formula, even with especially constructed difficult stiff systems. However only recent advances in more intelligent sparse matrix algorithms [2], and some way of event clustering, brought the expected speedup by bringing a constant time to solve a single event, and reducing the overhead.

To show the behavior of the presented method, results are presented below for an example circuit. The circuit has four pairs of analog differential inputs, two four-to-one analog multiplexers, a differential amplifier, and an eight bit AD converter, with the necessary control logic. The circuit is modeled with *both analog and digital components*, as these are both allowed as component primitives in our program. In the transient simulation four analog values are selected after each other and processed. The total simulation time is about 4 minutes on an Apollo DN3000 workstation, including input parsing and writing the output waveforms. Regarding the complexity of the circuit, the length of the simulation interval, and the limited performance of the workstation, this speed is impressive. During the analysis statistical values are gathered about latency aspects and cpu time spent in various parts of the program, to evaluate the simulation method:

dimension of matrix S	149
nonzero elements in L/U	431
number of dyadic updates	12380
average modified L/U elements per dyadic update	5.7
sparse F/B substitutions	17915
average size of $\Delta\dot{x}$	11.0
number of events processed	5158

times spent in:	
solving $\bar{x}$ and $\Delta\bar{x}$	30%
component operations for solving nonlinearities	21%
verifying LTEs and events	9%
printing output	7%
updating L/U decomposition	2.8%
others	30%

9. References

[1] Bennett J.M., "Triangular Factors of Modified Matrices", *Numerische Mathematik*, vol. 7, pp. 217-221, 1965

[2] Eijndhoven J.T.J., Stiphout M.T. van, "Latency Exploitation in circuit Simulation by Sparse Matrix Techniques", in: *Proc. Int. Symp. on Circuits And Systems*, Espoo, Finland, pp. 623-626, June 7-9, 1988

[3] Eijndhoven J.T.J., Jess J.A.G., "Mixed Mode Mixed Level Analysis with PWL Systems", in: *Proc. Int. Conf. on Circuits And Systems*, Montreal, Canada, pp. 1377-1380, may 7-10, 1984

[4] Gear C.W., Wells D.R., "Multirate Linear Multistep Methods", *BIT* (Denmark), vol. 24, no. 4, pp. 484-502, 1984

[5] Lelarasmee E., Ruehli A.E., Sangiovanni-Vincentelli A., "The waveform relaxation method for a time domain analysis of large scale integrated circuits", *IEEE Transactions on CAD of Integrated Circuits and systems*, vol. CAD 1, pp. 131-145, July 1982.

[6] Ortega J.M., Rheinboldt W.C., *Iterative Solution of Nonlinear Equations in Several Variables*, Academic Press, New York, 1970

[7] Palusinski O.A., "Simulation of Dynamic Systems Using Multirate Integration Techniques", *Trans. Soc. Comput. Simulation* (USA), vol. 2, no. 4, pp. 257-273, Dec. 1985

[8] Rodriguez G., Carrion F.J., "Stability Analysis for Multirate, Predictor-Corrector and Linear Multistep Integration Methods, Used in Real Time Simulation", *Proc. 1985 Summer Computer Simulation Conf.*, Chicago, Ill. (USA), pp. 371-376, July 22-24, 1985

[9] Sakallah K.A., Director S.W., "SAMSON2: An Event Driven VLSI Circuit Simulator", *IEEE Trans. on CAD*, vol. CAD-4, no. 4, pp. 668-684, Oct. 1985

[10] Skelboe S., "Stability properties of Backward Differentiation Multirate Formulas", *Applied Numerical Mathematics*, vol. 5, pp. 151-160, 1989

[11] Stiphout M.T., Eijndhoven J.T.J. van, Buurman H.W., "PLATO: A New Piecewise Linear Simulation Tool", in: *Proc. European Design Automation Conference*, Glasgow, G. Brittain, March 12-15, 1990 (This proceedings).

[12] White J. K., Sangiovanni-Vincentelli A., *"Relaxation Techniques for Simulation of VLSI Circuits"*, Kluwer Academic, 1987.