MASSACHUSETTS INSTITUTE OF TECHNOLOGY VLSI PUBLICATIONS

VLSI Memo No. 88-481
October 1988

ACCELERATING RELAXATION ALGORITHMS FOR CIRCUIT SIMULATION USING WAVEFORM-NEWTON AND STEP-SIZE REFINEMENT

R. A. Saleh, J. K. White, A. R. Newton, and A. L. Sangiovanni-Vincentelli

Abstract

A new relaxation algorithm for circuit simulation that combines the advantages of iterated timing analysis (ITA) and waveform-relaxation (WR) is described. The method is based on using an iterative step-size refinement strategy with a waveform-relaxation-Newton (WRN) algorithm. All three relaxation techniques, ITA, WR, and WRN, are compared and experimental results that indicate the strengths and weaknesses of the methods are presented. In addition, a new convergence proof for the waveform-Newton method for systems with nonlinear capacitors is provided. Finally, it is shown that the step-refined WRN algorithm can be implemented on a parallel processor in such a way that not only can different subsystems be processed in parallel but, in addition, the solution at different timepoints of the same subsystem can be computed in parallel.

Microsystems Research Center Room 39-321 Massachusetts Institute of Technology Cambridge Massachusetts 02139 Telephone (617) 253-8138

Acknowledgements

This work was supported in part by the Defense Advanced Research Projects Agency under contract numbers NESC-N39, and N00014-87-K-0825 and by NSERC of Canada, IBM Company, Phillips, Hewlett-Packard, Texas Instruments, and MICRO.

Author Information

Saleh: Coordinated Science Laboratory, Department of Electrical and Computer Engineering, University of Illinois, Urbana, IL.
White: Research Laboratory of Electronics and the Department of Electrical Engineering and Computer Science, Room 36-880, MIT, Cambridge, MA 02139. (617) 253-2543.
Newton and Sangiovanni-Vincentelli: Department of Electrical Engineering and Computer Science, University of California at Berkeley, Berkeley, CA 94720. (415) 642-6000.

Accelerating Relaxation Algorithms for Circuit Simulation Using Waveform-Newton and Step-Size Refinement

R. A. Saleh
Coordinated Science Laboratory
Department of Electrical and Computer Engineering
University of Illinois, Urbana, IL
J. K. White
Research Laboratory of Electronics and the
Department of Electrical Engineering and Computer Science
Massachusetts Institute of Technology, Cambridge, MA
A. R. Newton
A. L. Sangiovanni-Vincentelli
Department of Electrical Engineering and Computer Sciences
University of California at Berkeley, CA

July 20, 1988

Abstract

A new relaxation algorithm for circuit simulation that combines the advantages of Iterated Timing Analysis (ITA) and waveform relaxation (WR) is described. The method is based on using an iterative stepsize refinement strategy with a waveform-relaxation-Newton (WRN) algorithm. All three relaxation techniques, ITA, WR, and WRN, are compared and experimental results that indicate the strengths and weaknesses of the methods are presented. In addition, a new convergence proof for the waveform-Newton method for systems with nonlinear capacitors is provided. Finally, it is shown that the step-refined WRN algorithm can be implemented on a parallel processor in such a way that not only can different subsystems be processed in parallel but, in addition, the solution at different timepoints of the same subsystem can be computed in parallel.

1 Introduction

The implicit multistep integration algorithms used in general-purpose circuit simulation programs, like SPICE2[NAG75] and ASTAP[WEE73], have proven to be reliable but are computationally expensive when applied to large systems. This is because each step of the numerical integration requires the implicit solution of a large nonlinear algebraic system. If the circuit simulation program is intended for the simulation of mostly MOS digital circuits, then it is possible to exploit the properties of these types of circuits to improve the simulator's efficiency. In particular, the fact that MOS digital circuits can be partitioned into loosely or unidirectionally coupled subsystems can be exploited by using iterative decomposition algorithms, and the fact that the different nodes in an MOS digital circuits often change at very different rates can be exploited by using multirate integration techniques[GEA80].

A variety of algorithms have been applied to the simulation of MOS digital circuits that attempt to exploit its loosely coupled and multirate nature(for a comprehensive list of references see[NEW83] or [WHI86]). We will focus on three relaxation-based methods, the Iterated Timing Analysis (ITA) method used in the SPLICE programs [SAL83, KLE84, SAL87A], the waveform relaxation(WR) algorithm used in the RELAX programs[LEL82, WHI86], and a third new algorithm based on combining a waveform-Newton algorithm[WHI86, GUA85] with an iterative refinement timestep selection strategy.

Section 2 begins with a brief description of the classical circuit simulation algorithms used in programs like SPICE and ASTAP. In Section 3, the Iterated Timing Analysis and waveform algorithms are presented, along with a short discussion of their relative merits. In Section 4, the waveform relaxation-Newton (WRN) algorithm and the iterative refinement strategy[WHI85] for selecting the numerical integration timesteps are described, and the advantages of the combined algorithm on a parallel processor are mentioned. In Section 5, a brief description of the implementation of the relaxation algorithms is given, and experimental results that indicate the strengths and weaknesses of the three techniques are presented.

2 Classical Circuit Simulation

In this section we will describe the direct methods used in circuit simulation programs like SPICE[NAG75] or ASTAP[WEE73]. These methods are referred to as standard or classical methods, partly because SPICE and ASTAP are extensively used, and partly because in this area, any method more than ten years old is considered to be a classical method.

The behavior of a noninductive electrical circuit can be described by a system of equations involving node voltages, resistive currents, and capacitor charges. This system of equations can be constructed from a circuit description using the technique referred to as nodal analysis[DES69], which involves applying the Kirchoff Current Law (KCL) to each node in the circuit, and applying the constitutive or branch equations to each circuit element. Systems of equations so generated have the following form,

$$\frac{d}{dt}q(v(t), u(t)) = g(v(t), u(t)) \qquad (1)$$

where n is the number of circuit nodes excluding the reference, $q(v(t), u(t)) \epsilon \mathbb{R}^n$ is the vector of sums of capacitive charges at each node, $g(v(t), u(t)) \epsilon \mathbb{R}^n$ is the vector of sums of resistive currents at each node, $u(t) \epsilon \mathbb{R}^m$ is the vector of input voltages, and $v(t) \epsilon \mathbb{R}^n$ is the vector of node voltages.

It is possible to extend the nodal analysis technique to include circuits with inductors and voltage sources by using modified nodal analysis[HO75], while still preserving the form of (1)[WHI86]. The unknowns then become a mixture of node voltages and element currents, and the left-hand side of (1) becomes a vector of charges and inductor fluxes. In order to simplify the notation used in this paper, we will refer to the above nodal analysis form with only node voltages as the unknowns. However, many of the comments made in the following about these systems apply to any system in the form of (1).

The trapezoidal numerical integration algorithm is frequently used to approximate the system of (1) by a sequence of implicit algebraic equations. Given a timestep, h, the trapezoidal integration algorithm applied to (1) yields:

$$q(v(t+h), u(t+h)) - q(v(t), u(t)) = 0.5h[g(v(t+h), u(t+h)) + g(v(t), u(t))] \tag{2}$$

where $v(t)$ is known, and the equation must be solved to compute $v(t+h)$. The iterative Newton-Raphson algorithm is usually used to solve the implicit nonlinear algebraic system given by (2). The Newton-Raphson iteration equation for solving a general nonlinear system of the form $F(x) = 0$ is $J_F(x^k)(x^{k+1} - x^k) = -F(x^k)$ where $F(x^k)$ is referred to as the Newton residue and J_F is the Jacobian matrix of F with respect to x.

If the Newton algorithm is used to solve (2) for $v(t+h)$, the residue, $F(v^k(t+h))$, is:

$$F(v^k(t+h)) = q(v^k(t+h), u(t+h)) - q(v(t), u(t)) - 0.5h(g(v^k(t+h), u(t+h)) + g(v(t), u(t))) \tag{3}$$

and the Jacobian of $F(v^k(t+h))$, $J_F(v^k(t+h))$ is:

$$J_F(v^k(t+h)) = \frac{\partial q(v^k(t+h), u(t+h))}{\partial v} - 0.5h\frac{\partial g(v^k(t+h), u(t+h))}{\partial v}. \tag{4}$$

Then $v^{k+1}(t+h)$ is derived from $v^k(t+h)$ by solving the linear system of equations

$$J_F(v^k(t+h))[v^{k+1}(t+h) - v^k(t+h)] = -F(v^k(t+h)) \tag{5}$$

using some form of Gaussian elimination. The Newton iteration is continued until sufficient convergence is achieved, that is $\|v^{k+1}(t+h) - v^k(t+h)\| < \epsilon$ and $F(v^k(t+h))$ is close enough to zero.

3 Relaxation In Circuit Simulation

As is clear from (5), using classical techniques to simulate large circuits implies solving large linear systems, usually with some form Gaussian elimination. In general, matrix solution by Gaussian elimination grows in complexity *like* n^3 where n is the number of unknowns in the system. This rapid growth in complexity has focused attention on finding other approaches for solving large linear systems. Since the matrices that describe the linear systems in circuit simulation problems are usually sparse, that is, most of matrix entries are zero, clever data structures and careful reordering algorithms have been used to reduce the complexity of Gaussian elimination for sparse matrices to less than n^2[KUN86]. However, sparse matrix techniques based on Gaussian elimination still grow superlinearly with the number of unknowns, and therefore researchers have continued to seek matrix solution methods that do not grow as rapidly.

Relaxation methods comprise one class of iterative matrix solution methods that tend to have a slower computational growth than sparse Gaussian elimination. In particular, the computation per iteration grows linearly with the number of nonzero matrix entries. The disadvantage of relaxation is that each iteration only produces an *approximate* solution to a linear system, and repeated relaxation iterations do not necessarily improve the approximation. For circuit simulation problems, relaxation methods work well because, in general, repeated relaxation iterations do converge to the exact solution[NEW83]. As an example, consider solving the linear problem $Ax - b = 0$ where $x = (x_1, ..., x_n)^T, b = (b_1, ..., b_n)^T, x_i, b_i \epsilon \mathbb{R}$, and $A = (a_{ij}), A \epsilon \mathbb{R}^{n x n}$. One could attempt to solve this problem by solving the equations one row at time, guessing values for the x_i's that have not been computed. This leads to Algorithm 1, the Gauss-Seidel relaxation algorithm, which will be referred to several times in the following text.

Algorithm 1 - Gauss-Seidel Algorithm for solving $Ax - b = 0$.

```
*The superscript $k$ is the iteration count and $\epsilon$ is a small
positive number.*
$k \leftarrow 0$.
Guess some $F^0$.
**repeat** {
    $k \leftarrow k + 1$.
    **for each** $(i \epsilon \{1, .., n\}) x_i^k = \frac{1}{a_{ii}}[b_i - \sum_{j=1}^{i-1} a_{ij} x_i^k + \sum_{j=i+1}^{n} a_{ij} x_i^{k-1}]$.
} **until** $(\|x^k - x^{k-1}\| \leq \epsilon)$
```

The linear Gauss-Seidel relaxation in Algorithm 1 could be used instead of Gaussian elimination to solve the linear matrix problem of (2), and this might reduce the computation time for the matrix solution in a circuit simulation program. However, if the circuit being simulated has fewer than several thousand nodes, at least half of the computation time is spent evaluating the Jacobian and the Newton residue. For this reason, other relaxation schemes have been applied to circuit simulation problems, ones that attempt to both avoid Gaussian elimination and reduce the time to evaluate the Jacobian and Newton residue.

Historically, the first approach to using relaxation in a circuit simulation program was to substitute a relaxation method for the Newton method to solve the implicit nonlinear algebraic systems generated by the discretization method. Early programs based on this approach attempted only to produce approximate results, and performed only one relaxation iteration for each timestep. The techniques used in these programs were referred to as "timing analysis", and when the relaxation iteration was carried to convergence to produce accurate results, the algorithm was referred to as Iterated Timing Analysis (ITA) [CHA75, NEW83]. A second, more recently developed approach, applied the relaxation directly to the differential equation system, replacing solving a large differential equation system with solving a collection of small differential equation subsystems[LEL82]. This second approach is referred to as waveform relaxation(WR) because the iterates are functions or waveforms over the simulation interval. In the following two subsections we describe the ITA and WR approaches in more detail.

3.1 Iterated Timing Analysis

The above linear Gauss-Seidel relaxation "recipe" can be applied to directly to solving the nonlinear algebraic system $F(v(t+h)) = 0$ given in (3). The nonlinear system can be rewritten as $F(v(t+h)) = (f_1(v(t+h))$ $,\ldots, f_n(v(t+h)))^T$ where $f_i : \mathbb{R}^n \rightarrow \mathbb{R}$. Following the Gauss-Seidel approach for linear systems, at each step of the relaxation, the $v_i(t+h)$ element is updated by solving the implicit algebraic equation

$$f_i(v^{k+1,i}(t+h)) = 0 \tag{6}$$

for v_i^{k+1}, where $v^{k,i} \equiv (v_1^k, \ldots, v_i^k, v_{i+1}^{k-1}, \ldots, v_n^{k-1})^T$.

It is possible to use the Newton-Raphson algorithm to solve the implicit algebraic system of (6) accurately at each step, and algorithms so generated are referred to as relaxation-Newton methods. However, in this application, the Newton iteration is not usually carried to convergence as it has been shown that the rate of convergence of nonlinear relaxation is not reduced if, rather than solving the implicit algebraic systems at each step, only one iteration of the Newton method is used [ORT70]. Such a Gauss-Seidel-Newton algorithm applied to systems of the form $F(v) = 0$ is (using the notation of (6)),

$$v_i^{k+1}(t+h) = v_i^k(t+h) - [\frac{\partial f_i(v^{k+1,i-1}(t+h))}{\partial v_i}]^{-1} f_i(v^{k+1,i-1}(t+h)). \tag{7}$$

Applied to the system in (3) we get

$$v_i^{k+1} = v_i^k -$$

$$\frac{q_i(v^{k+1,i-1}(t+h),u(t+h)) - q_i(v(t),u(t)) - 0.5h(g_i(v^{k+1,i-1}(t+h),u(t+h)) + g_i(v(t),u(t)))}{\frac{\partial q_i(v^{k+1,i-1}(t+h),u(t+h))}{\partial v} - 0.5h\frac{\partial g_i(v^{k+1,i-1}(t+h),u(t+h))}{\partial v}} \quad (8)$$

The following convergence theorem is a minor modification of the result in [ORT70,WHI86].

Theorem 1 *Let the Gauss-Seidel-Newton relaxation algorithm be used to solve for $v(t+h)$ in (3). If $g(v,u)$ is continuously differentiable with respect to v uniformly in u, $u(t)$ is continuous with respect to t, $\frac{\partial q}{\partial v}(v,u)$ is strictly or irreducibly diagonally dominant uniformly over all v and u, and $v(t) + O(h)$ (where $O(h)$ is continuous in h and $\lim_{h \to 0} O(h) = 0$) is used as the starting point for the relaxation, then there exists an* h *such that for all $h_m \leq$* h *the relaxation converges to the solution of (3)* ■

The relaxation-Newton methods are popular for circuit simulation problems for two reasons. The first is that for a broad class of circuits the capacitance matrix, given as $\frac{\partial q}{\partial v}(v,u)$, is strictly diagonally dominant. This is the case if the circuit contains two-terminal positive capacitors (linear or nonlinear), or any other elements whose charge function has a diagonally dominant Jacobian with positive diagonal entries, and there exists some nonzero capacitance to ground or a voltage source at each node in the circuit. Therefore, for these circuits, the relaxation-Newton algorithms are guaranteed to converge if the integration timestep is made small enough.

The second reason for the popularity of relaxation-Newton methods is that with proper application they can be used both to avoid solving a large matrix problem by Gaussian elimination and to avoid computing portions of the Newton residue and the Jacobian. As can be seen in from (7), only the diagonal terms of the Jacobian, $\frac{\partial f_i}{\partial v_i}$, need be computed. In addition, the computation of the Newton residue and the Jacobian can be further reduced using two bypassing schemes [SAL83] that skip the computation of portions of the Jacobian and the residue. One type of bypassing is implemented by noting whether the components of v on which f_i depends have changed significantly in a relaxation-Newton iteration, and if none of them have, then f_i is not re-evaluated. Bypassing in time can also be implemented by checking if $\dot{v}_i$ is close enough to 0 and, if so, then $v_i(t+h)$ will be equal to $v_i(t)$ and therefore v_i need not be recomputed at $t+h$. Note that the effect of bypassing in time is to exploit *waveform latency*, that is, if a node in the circuit is not changing, then the contributions to the Jacobian and Newton residue of the devices connected to that node need not be computed at subsequent points in time.

3.2 Waveform Relaxation

The classical approach and the ITA approach without latency exploitation suffer a loss of efficiency because they apply the integration method to the entire system simultaneously. This forces every differential equation in the system to be discretized identically, and this discretization must be fine enough so that the fastest-changing state variable in the system is accurately represented. If it were possible to pick different timesteps for each differential equation in the system, so that each could use the largest timestep that would accurately reflect the behavior of its associated state variable, then the efficiency of the simulation would be greatly improved. This is referred to as the multirate problem[GEA80], and numerical integration methods that allow for different state variables to use different timesteps are called multirate integration methods.

Circuit simulation programs that use ITA with latency exploitation do exhibit some of the characteristics of a multirate integration method. If, at a given timestep, the v_i variable is at its equilibrium (or stationary) point, and the v_j variables on which v_i depends do not change, then v_i will retain the value it had before the timestep. In fact, v_i will never be recomputed until some v_j on which it depends changes. If v_i is bypassed for several timesteps the effect is the same as if a large timestep were used to compute v_i. Therefore, a bypassing algorithm exploits the kind of multirate behavior that stems from a system in which most of of the variables remain at equilibrium. The ITA approach with latency exploitation does not, however, take full advantage of a system for which the state variables have different rates of motion but are not at equilibrium.

It is possible to create a full multirate integration method for solving (1) by applying the Gauss-Seidel recipe for relaxation directly to the differential equation system *before* introducing discrete approximations. If such an approach were applied to (1) then at each step of the relaxation, the *waveform* $v_i(t)$ for all $t \epsilon [0,T]$ is updated by solving the differential iteration equation

$$\frac{d}{dt} q_i(v^{k+1,i+1}(t), u(t)) - g_i(v^{k+1,i+1}(t), u(t)) = 0 \quad v_i^{k+1}(0) = v_{i0} \tag{9}$$

for $v_i^{k+1}(t)$ on $[0,T]$. In this approach, each differential equation in system is solved independently, and each can use an independent set of integration timesteps. Therefore, the waveform relaxation algorithm is inherently a multirate integration method.

The convergence of WR is guaranteed for most examples of practical interest by the following theorem (1) [LEL82, WHI86].

Theorem 2 *If, in a system of the form of (1), $g(v,u)$ is Lipschitz continuous with respect to v uniformly in $u, u(t)$ is continuous with respect to t and $\frac{\partial g}{\partial v}(v,u)$ is strictly or irreducibly diagonally dominant uniformly over all v and u, then the Gauss-Seidel waveform relaxation algorithm applied to such a system will converge to the solution to the system* ■

Theorem 2 guarantees the convergence of WR for the continuous case, but does not directly apply if a discretization integration formula is used to solve for the iteration waveforms. It is possible to prove a convergence theorem for the *discretized* WR algorithm, but only by assuming some conditions on the timesteps. In particular, one can show that if multistep methods are used to solve the iteration equations, then the discretized WR algorithm converges provided the integration timesteps are smaller than some bound[NEV86]. Not surprisingly, the bound on the integration timesteps to insure WR convergence is similar to the bound on the timesteps required to insure the ITA algorithm above converges. In fact, for linear systems and nonmultirate timesteps, the timestep constraints for ITA and waveform relaxation are identical[WHI86].

4 Waveform-Relaxation-Newton Algorithm

The advantage of the standard WR algorithm is that it fully exploits multirate behavior by solving the differential equations in a decoupled fashion. However, each iteration involves solving nonlinear differential equations accurately, and is therefore computationally expensive. Although ITA can not fully exploit multirate behavior easily[SAL87A], it is, in many cases, more efficient than standard WR. This is partly because each relaxation iteration for the ITA method is much cheaper, as the nonlinear relaxation iteration equations are solved approximately with a single Newton iteration. Perhaps it is the obvious next step to try a similar approach with the standard WR algorithm. That is, use a single iteration of some kind of waveform-Newton algorithm to approximately solve the WR iteration equations.

It is reasonably straight-forward to derive the waveform-relaxation-Newton (WRN) algorithm [GUA83, BOK83] and show that WRN has similar convergence properties to standard WR. In addition, the iteration equations for WRN are time-varying linear differential equations and are easier to solve than the nonlinear differential iteration equations of WR. However, WRN does not prove to be much more efficient than WR when applied to simulating most digital circuit examples, because whereas WR usually converges in fewer than five iterations, and WRN may take many more.

The fact that the WRN algorithm may take many iterations to converge makes it possible to improve the algorithm's efficiency by solving the iteration equations crudely at first, and then increasing the accuracy with each iteration. With WRN, it is particularly efficient to use coarse timesteps in the early iterations, and then refine as the iterations approach convergence. Note that such a technique is not so helpful with WR because WR usually converges so rapidly that there are not enough iterations to do meaningful refinement. Also, the Newton method used at each timestep in the WR iterations may not converge if very large timesteps are used.

In this section, a WRN algorithm that incorporates timestep refinement is described. In Section 4.1 the waveform-Newton (WN) [KAN64] is derived and some of its limitations are described. In Section 4.2, WN is combined with WR to produce the waveform-relaxation-Newton (WRN) algorithm. The iterative timestep refinement strategy used with WRN is described in Section 4.3, and its suitability for parallel computation is mentioned in Section 4.4.

4.1 Derivation of the Waveform Newton Algorithm

In order to derive the waveform-Newton algorithm it is helpful to think in general terms. Finding a solution to (1) can be thought of as finding a v such that $F(v) = 0$, where $v \in \omega$, the space of continuous and differentiable functions that map the interval $[0,T]$ into $\mathbb{R}^n$, and F is a function derived from (1) that takes $\omega \rightarrow \omega$. The iteration equation for the Newton-Raphson algorithm for finding a solution v such that $F(v) = 0$ is,

$$J_F(v^k)[v^{k+1} - v^k] = -F(v^k) \tag{10}$$

where $J_F(v)$ is the Frechet derivative of $F(v)$ with respect to v. The Frechet derivative is defined [ORT70] as the linear operator on ω such that

$$\lim_{\delta \to 0} \frac{1}{||\delta||} ||F(v+\delta) - F(v) - J_F(v)\delta|| = 0. \tag{11}$$

As mentioned above, the function F is derived from (1) so that $F(v) = 0$ implies that v solves (1). A pointwise in t description of F is then given by

$$(F(v))(t) = \frac{d}{dt}q(v(t), u(t)) - g(v(t)). \tag{12}$$

This pointwise description of F, and the definition of the Frechet derivative given above, can be used to evaluate the derivative of F, denoted as J_F. A convenient norm on ω in which to evaluate $J_F(v)$ from (11) and (12) is

$$||v|| = ||v(0)||^{\mathbb{R}^n} + max_{\tau \epsilon [0,t]} ||\dot{v}(\tau)||^{\mathbb{R}^n}, \tag{13}$$

where $v \epsilon \omega$, $\dot{v}(t)$ is the usual derivative of $v(t)$ with respect to t, and $|| \, ||^{\mathbb{R}^n}$ is any norm on $\mathbb{R}^n$.

To begin, consider that

$$\begin{aligned} (F(v+\delta))(t) - (F(v))(t) &= \tfrac{d}{dt}[q(v(t)+\delta(t), u(t)) - q(v(t), u(t))] - [g(v(t)+\delta(t), u(t)) - g(v(t), u(t))] \\ v(0) = v_0 \quad \delta(0) = 0. \end{aligned} \tag{14}$$

Expanding and approximating to order $(||\delta||^2$,

$$(F(v+\delta))(t) - (F(v))(t) = \frac{d}{dt}[\frac{\partial q(v(t), u(t))}{\partial v}\delta(t)] - \frac{\partial f(v(t))}{\partial v}\delta(t) + O(||\delta||^2) \tag{15}$$

From the definition of the Frechet derivative, (15) implies that

$$(J_F(v)\delta)(t) = \frac{d}{dt}[\frac{\partial q(v(t), u(t))}{\partial v}\delta(t)] - \frac{\partial g(v(t), u(t))}{\partial v}\delta(t). \tag{16}$$

Substituting the computed derivative (16) and F given by (12) into (11) and rearranging yields

$$\frac{d}{dt}[q(v^k(t), u(t)) + \frac{\partial q(v^k(t), u(t))}{\partial v}\delta^{k+1}(t)] = g(v^k(t), u(t)) + \frac{\partial g(v^k(t), u(t))}{\partial v}\delta^{k+1}(t). \tag{17}$$

where $\delta^{k+1}(t) = v^{k+1}(t) - v^k(t)$. Note that if $v^k(0) = v_0$ then $\delta(0) = 0$. We will refer to (17) as the waveform-Newton(WN) algorithm for solving (12).

The above formal derivation of the WN algorithm has a very simple interpretation. The differential equation is linearized about an initial guess waveform whose value at time zero matches the given initial condition for the differential equation. Then, the guess waveform is updated by solving the resulting linearized differential equation (with an initial condition of zero). The original differential equation is then relinearized about the updated guess, and the process repeated until convergence is achieved.

As WN is just the function-space extension of the classical Newton-Raphson algorithm, it will converge quadratically when the iterated value is close to the correct solution[KAN64]. The WN algorithm also has the attractive property that it converges globally when applied to equations of the form of (1), given mild assumptions on the behavior of the charge and current functions, $q(v)$ and $g(v)$, and provided the initial guess waveform matches the initial condition for the differential equation. In particular, we have the following theorem about the convergence of the WN algorithm whose proof is given in the appendix.

Theorem 3 *For any system of the form of (1) in which $\frac{\partial q}{\partial v}$ is differentiable, Lipschitz continuous, and has a uniformly bounded inverse with respect to v for all u; g is differentiable and Lipschitz continuous; and $v^0(t)$ is a continuous, differentiable function such that $v^0(0) = v_0$; then the sequence of waveforms, $\{v^k\}$, generated by the WN algorithm converges uniformly to the solution of (1)* ∎

The iteration equations for the WN algorithm are time-varying linear differential equations that are easier to solve numerically than the original system of nonlinear differential equations. For example, if WN iteration equations are solved with the same discretization techniques as used for the classical direct method, then since the equations are linear, only a single matrix solution need be performed at each timepoint. Also, linear time-varying systems can be solved with a variety of efficient numerical techniques other than the standard discretization methods, such as collocation and spectral methods[GUA83].

Note that the discretized WN algorithm is as ill-suited to the simulation of large problems as the classical direct methods, because it still requires the solution of large linear equation systems, and can not easily

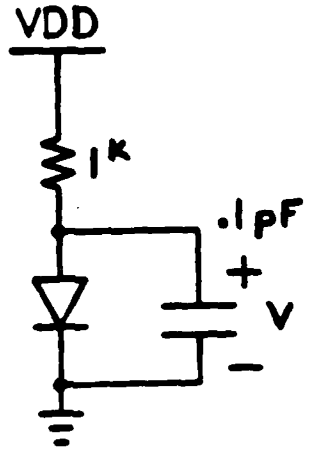

Figure 1: Resistor-Diode Example

exploit multirate behavior. But as the discretized WN algorithm and the classical direct methods are of such a similar nature, it is perhaps useful to compare their relative efficiencies. Clearly, for linear problems, the classical direct and the discretized WN methods are identical. However, the discretized WN can be much less efficient than classical direct methods when used to simulate circuits containing highly nonlinear elements such as diodes. This is due to the difference in how the classical and WN methods get the initial guesses for their respective Newton methods. In the classical approach, generating an initial guess for the Newton method implies projecting the behavior of the system forward by one timestep. In WN, one projects the behavior of the system forward for an entire waveform. Therefore, in the classical method the initial guess is almost certainly in or near the region of quadratic convergence for the Newton method, whereas that would rarely be true for WN.

To demonstrate this difficultly with WN, consider the problem illustrated in Fig. (1), a simple resistor-diode circuit with a grounded capacitor. Fig. (2) shows the waveform iterations obtained using WN to solve the circuit, given an initial guess of $v^0(t) = 0$ for all $t \epsilon [0,1]$. Note that the first computed waveform $v^1(t)$ is quite far from the correct solution, and subsequent iterations move very slowly back to the correct solution. This slow convergence is common when applying Newton methods to exponential nonlinearities, given a poor initial guess. In this case, over 50 iterations are necessary to achieve satisfactory convergence.

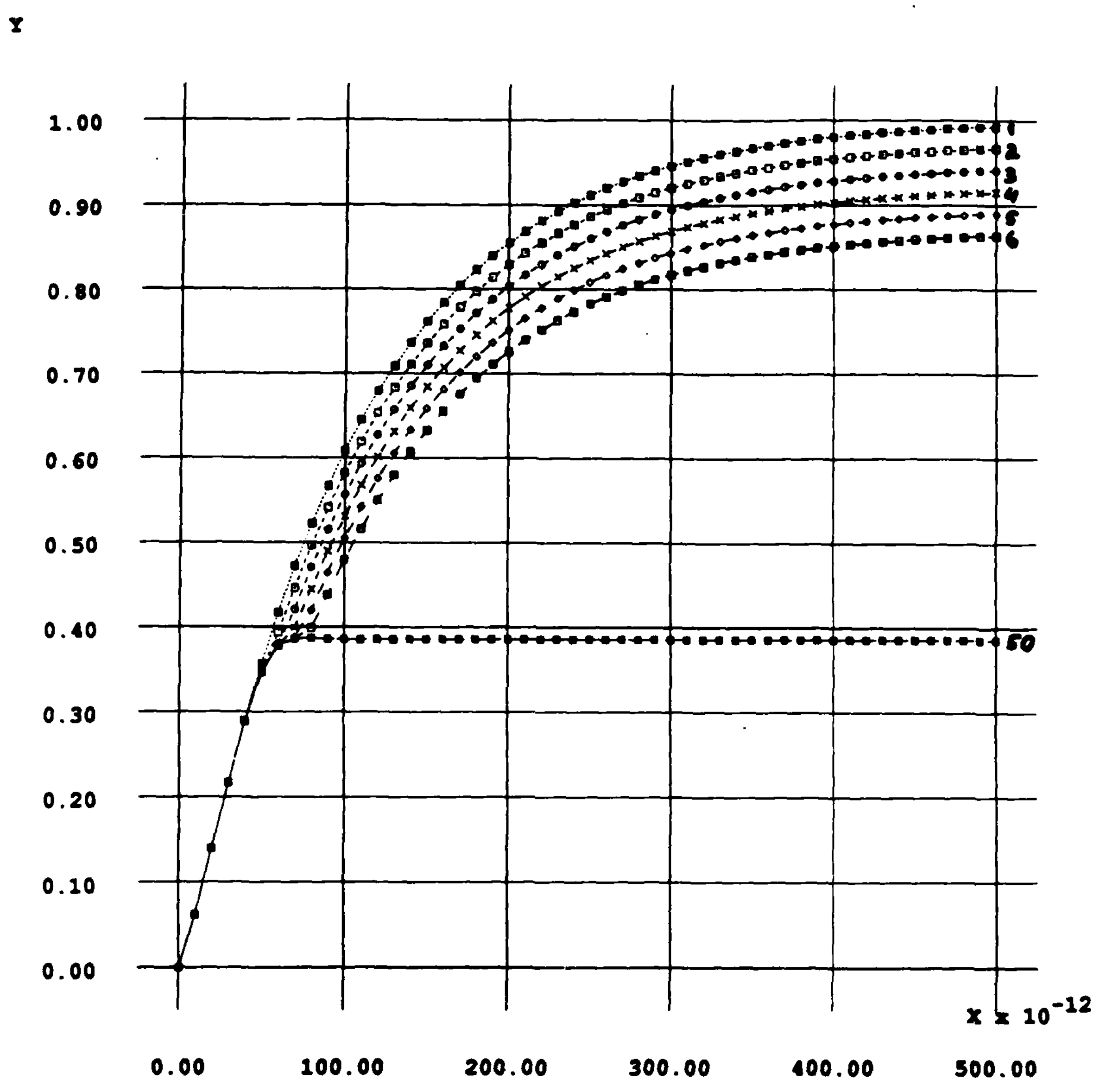

Figure 2: Waveform Newton Iterations for the Resistor-Diode Example

A wide variety of limiting techniques can be used to improve the convergence of WN in this case. For example, the change in the waveform from one iteration to the next could be limited, in a way analogous to step limiting in the standard algebraic Newton method[NAG75]. There is a well-known strategy that is particularly useful for MOS circuits, where the only diodes in the circuit are associated with the source-to-substrate and drain-to-substrate junctions, and are usually reverse-biased. Instead of using the correct derivative of the diode current with respect to the voltage across the diode, the derivative can be approximated by the diode current divided by the voltage across the diode, an approximation known as "line-through-the-origin". Since only the Jacobian is altered, if the so created pseudo-Newton method converges, it converges to the correct result. The danger is that if the diode is substantially forward biased, an unlikely event for MOS circuits, this pseudo-Newton method may not converge no matter how close the initial guess.

4.2 The Waveform Relaxation Newton Algorithm

One useful application of the WN algorithm, and the main focus in this paper, is to combine it with the WR algorithm to construct the waveform extension of the relaxation-Newton algorithms presented in Section 3.1, as in Algorithm 2 below.

Algorithm 2 - (WRN Gauss-Seidel Algorithm)

```
The superscript k denotes the iteration count, the subscript $i \epsilon \{1, ..., N\}$
denotes the component index of a vector and $\epsilon$ is a small positive number.
$k \leftarrow 0$.
Guess waveform $v^0(t); t\epsilon[0,T]$
such that $v^0(0) = v_0$ (for example, set $v^0(t) = v_0, t\epsilon[0,T]$);
repeat {
    $k \leftarrow k + 1$
    for all ($i$ in $N$) {
        solve
            $\frac{d}{dt}[q_i(v^{k+1,i}(t), u(t)) + \frac{\partial q_i(v^{k+1,i}(t), u(t))}{\partial v_i}(v_i^{k+1}(t) - v_i^k(t))]$
            $-[f_i(v^{k+1,i}(t), u(t)) + \frac{\partial f_i(v^{k+1,i}(t), u(t))}{\partial v_i}(v_i^{k+1}(t) - v_i^{k+1}(t))] = 0$
        for $(v_i^{k+1}(t); t\epsilon[0,T])$, with the initial condition $v_i^{k+1}(0) = v_{i0}$.
    }
} until $(\|v^{k+1} - v^k\| \leq \epsilon)$
```

In the waveform relaxation-Newton algorithm (WRN) [BOK83, GAU83, WHI86] above, the WR iteration equations are solved approximately by performing one step of the waveform-Newton method with each waveform relaxation iteration. This is analogous to the single Newton iteration strategy of the nonlinear relaxation methods used in Iterated Timing Analysis. And like the WR algorithm, each equation in Algorithm 2 is a differential equation in one unknown variable v_i^k, but in this case the nonlinear differential iteration equations have been replaced by simpler time-varying linear differential equations.

Given the global convergence properties of both the original WR and the WN algorithms, it is not surprising that the WRN algorithm has global convergence properties, and the proof for this is quite similar to the proof of the WR and WN convergence theorems.

Theorem 4 *If the assumptions in Theorem 2 and Theorem 3 are both satisfied, then the sequence* $\{v^k\}$ *generated by the WRN algorithm converges to the solution of (1) on any bounded interval* $[0,T]$ ■

4.3 Timestep Control Strategy for WRN

As mentioned above, the amount of computation performed in the early iterations of the WRN can be reduced by using coarse numerical integration timesteps to solve the differential relaxation equations initially, and then refining the timesteps as the iterations progress. Specifically, the first relaxation iteration is computed with a user-supplied maximum allowed numerical integration timestep. For subsequent relaxation iterations, the integration timesteps are chosen to be the same as those in the previous iteration unless the a *posteriori* local truncation error estimates for the timestep from the previous iteration is too large. In that case, half the previous iteration timestep is used.

To demonstrate this idea, consider the sequence of waveforms in Fig. 3. For the first iteration the maximum timestep is used, which produces the waveform in Fig. (3a), In this case, the computed LTE at T is too large, and therefore on the second iteration, the window interval is divided in half and two time-steps are taken, as shown in Fig. (3b). In the second iteration, the LTE is too large at time point T/2 but acceptable at time point T. Therefore, on the third iteration, only the first half interval is divided, as shown in Fig. (3c).

The iterative refinement strategy has the advantage that, in general, the timesteps will be placed more efficiently to control truncation error than if the standard predicted truncation error criteria is used. This is because the timestep selection is based only on more accurate a *posteriori* error estimates available from previous relaxation iterations. However, there are situations where too many time points will be placed. In particular, if, in some region of time, a "wavefront" moves through as the iterations progress (see Fig. (4)),

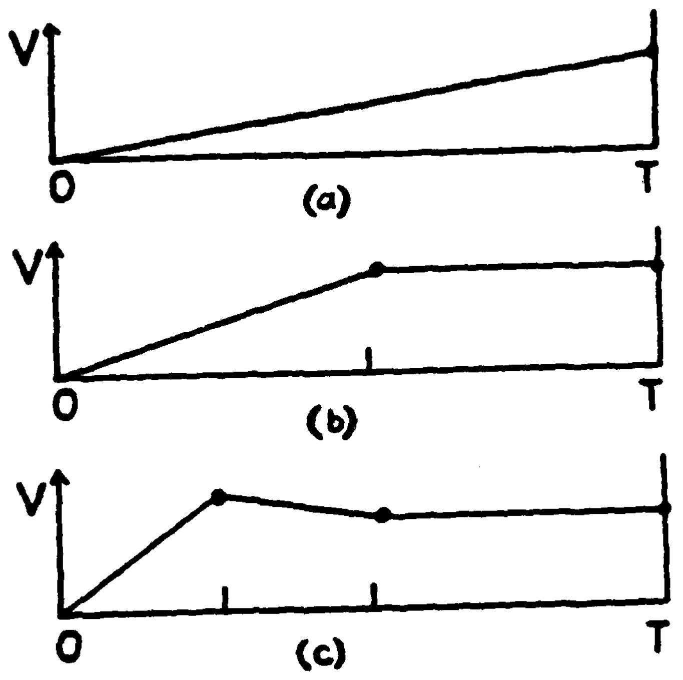

Figure 3: WRN Time-Step Control.

many timesteps will be placed in the wavefront's path. A way to overcome this problem is to *remove* time points on subsequent iterations if the LTE is small at a time point. This is not as simple an optimization as would seem, as such an approach can be unstable. That is, time points which are added on one iteration are removed on the next iteration, and then added again on the third iteration, etc.

4.4 Parallelizing WRN with Timestep Refinement

The WRN algorithm combined with the timestep refinement strategy has several advantages that make it a good algorithm for use on parallel processors[SAL87B]. As with any decomposition method, the decomposed subsystems can be solved independently on parallel processors[WHI86]. And as with any waveform relaxation method, solving the decomposed subsystems is a significant computation involving numerically integrating the independent differential equations over some interval of time. Therefore, in comparison, communication overhead is negligible.

The above WRN algorithm has an additional advantage for parallel computation. As the discretizations times are selected by a timestep refinement strategy, they are therefore known *a priori*, before beginning the calculation of the next iteration waveform. And since the iteration update equations for WRN are based on linearization about a previous iteration waveform, once the discretization times are known, the

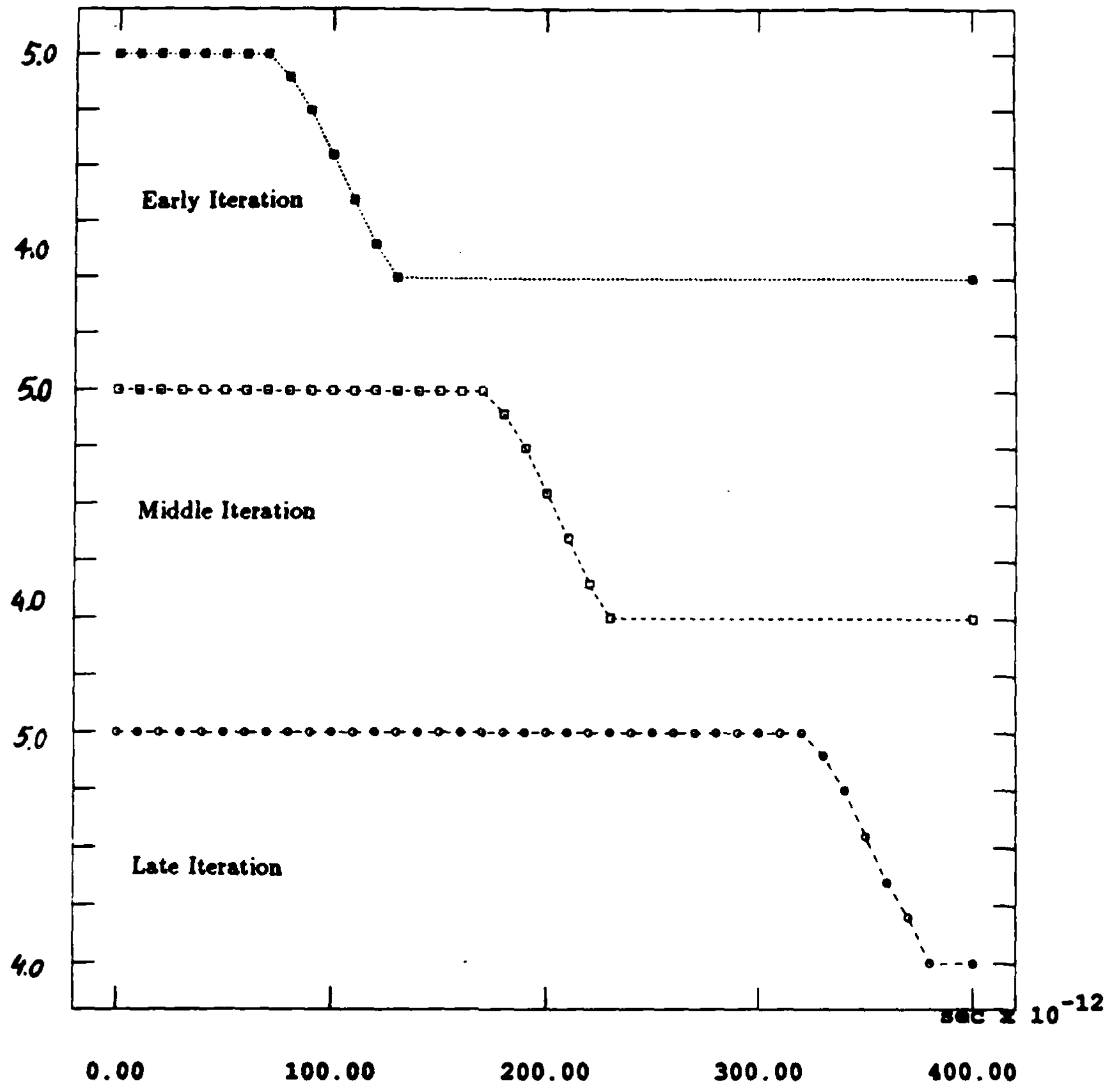

Figure 4: Wavefront Propagation During Iterations

linearized system at each of the discrete times can be computed in parallel. The result is that not only can the decomposed subsystems be computed in parallel, but most of the computation for each of the timesteps for each of the decomposed subsystems can be computed in parallel.

In order to see how the timestep parallelism can be exploited, consider using WN to solve the simple equation

$$\dot{x}(t) = f(x(t)) \quad x(0) = x_0, \tag{18}$$

where $x(t) \in \mathbb{R}^n$, and $m : \mathbb{R}^n \rightarrow \mathbb{R}^n$. The WN iteration equation is then

$$\dot{x}^{k+1}(t) = f(x^k(t)) + \frac{\partial f(x^k(t))}{\partial x}(x^{k+1}(t) - x^k(t)) \qquad x^{k+1}(0) = x^k(0) = x_0 \tag{19}$$

where $x^k(t)$ is the k^{th} Newton iterate. Discretizing (19) with backward-Euler leads to a sequence of M linear equations, where M is the number of timesteps. The equation that must be solved to compute $x^{k+1}(\tau_j)$, where τ_j is the time of the j^{th} timestep, is given by

$$\frac{1}{\tau_j - \tau_{j-1}}(x^{k+1}(\tau_j) - x^{k+1}(\tau_{j-1})) = f(x(\tau_j)) + \frac{\partial f(x^k(\tau_j))}{\partial x}(x^{k+1}(\tau_j) - x^k(\tau_j)). \tag{20}$$

Reorganizing (20) leads to

$$[\frac{1}{\tau_j - \tau_{j-1}}I - \frac{\partial f(x^k(\tau_j))}{\partial x}](x^{k+1}(\tau_j)) = \frac{1}{\tau_j - \tau_{j-1}}x^{k+1}(\tau_{j-1}) + f(x(\tau_j)) - \frac{\partial f(x^k(\tau_j))}{\partial x}x^k(\tau_j), \tag{21}$$

where I is the identity matrix in $\mathbb{R}^n$. The parallelism that can be exploited by simultaneously evaluating M timesteps can be seen by examining (21). Clearly, the evaluation and LU decomposition of the matrix on the left-hand side of (21), and the computation of all the terms on the right-hand side of (21) except the $\frac{1}{\tau_j - \tau_{j-1}}x^{k+1}(\tau_{j-1})$ term can be performed for all M timesteps simultaneously.

Once the parallelizable portion of (21) is complete, there is still some computation that must be performed serially to update the $x^{k+1}(\tau_j)$ values. This serial computation involves a matrix backsolve to compute $x^{k+1}(\tau_1)$, which is then used to complete the right-hand side for the equation for $x^{k+1}(\tau_2)$, from which $x^{k+1}(\tau_2)$ can be computed with a matrix backsolve, which is then used to complete the right-hand side of the equation for $x^{k+1}(\tau_3)$, etc. In total, this serial section involves M matrix backsolves and $M-1$ multiplication and additions to complete the right-hand sides. However, the backsolves are by far the smallest part of the timestep calculation, so this serial section is not too computationally expensive, and therefore processing timesteps in parallel can be effective.

5 Program Implementation and Simulation Results

In this section, we compare the basic algorithms used in three programs, SPLICE3.1, based on ITA, RELAX2, based on waveform relaxation, and a new program, SPLAX, based on WRN with iterative timestep refinement. The two aspects common to the three relaxation algorithms, that of partitioning a large problem into loosely coupled subcircuits and ordering the subcircuits for the relaxation, is presented in the next subsection. Then, the aspects where the three programs differ are described. In the last subsection, the comparison results are presented and interpreted.

5.1 Partitioning and Ordering

Relaxation algorithms have been applied in several programs for the simulation of MOS integrated circuits[NEW83, DUM86]. These programs do not use the relaxation algorithms exactly as described in earlier sections, but employ two important modifications to improve the speed of relaxation convergence. MOS circuits tend to have small subsystems that are very tightly coupled. In order to avoid the slow relaxation convergence that would result from trying to decompose such subsystems into single equations in one unknown, the circuit is decomposed into blocks. The blocks are solved using some form of Gaussian elimination, and the relaxation is applied to the block-decomposed system. Another feature of MOS circuits is that they tend to be directional. That is, it is possible to order the equations such that the system is almost lower-triangular, and finding this ordering can accelerate the convergence of the Gauss-Seidel relaxation.

Grouping together tightly coupled equations or blocks in a system, is referred to as *partitioning* the system. The partitioning algorithm used for the above three programs is based on trying to directly estimate the coupling between pairs of equations to determine which equations should be grouped together[WHI86]. In particular, the algorithm amounts to removing the coupling between two nodes, and then for each node replacing the remaining circuitry connected to each of the nodes by Norton equivalent conductances to ground. The coupling is then added back to the circuit, yielding a two node problem. The speed at which a relaxation algorithm applied to the two node problem will converge can be computed exactly, and if the speed is too slow, the two nodes are placed in the same partition or subcircuit.

Once the system has been partitioned, the resulting blocks are ordered so that the relaxation is applied to a problem that is as *block* lower triangular as possible. In some sense, partitioning and ordering the subsystem of equations are performing similar functions. They are both attempting to eliminate slow relaxation convergence due to two nodes in a large circuit being tightly coupled. There is, however, a *key* difference. If, for example, v_i is strongly dependent on v_j, *and* v_j is strongly dependent on v_i, then a partitioning algorithm should lump the two nodes together into one subsystem. However, if v_i is strongly dependent on v_j, but v_j is *weakly* dependent on v_i, then node i and node j should not be lumped together, but the ordering algorithm should insure that the system is block lower triangular by ordering the equations so that v_j is computed before computing v_i.

Resistors and capacitors do not exhibit the kind of unidirectional coupling that is of concern to the ordering algorithm. In fact, the only element type of concern to the ordering algorithm is the transistor, because it exhibits unidirectional coupling. That is, the drain and source terminals of an MOS transistor are strongly dependent on the gate terminal of the transistor, but the gate is almost independent of the drain and source. Therefore, the ordering algorithm used in our three programs attempts to order subsystems so that a subsystem containing a given transistor's drain or source is solved after the subsystem containing the given transistor's gate. If the circuit contains global feedback loops, this may not always be possible, and such loops are broken in a somewhat *ad hoc* fashion.

5.2 Differences Between Programs

In order to make meaningful comparisons between the effectiveness of the three algorithms, we tried to keep algorithms used in the programs as close to the same as possible. For example, all three programs are written in *C*, and as mentioned above, all three programs use the same ordering and partitioning algorithms. However, the nature of the three algorithms do force certain differences between the three programs.

Both SPLAX and RELAX2 use waveform relaxation-based algorithms and, because of the nonuniform way in which WR converges, it is much more efficient if the simulation interval $[0,T]$ is broken into subintervals, or windows, $[0,t_1],[t_1,t_2]$, ... , $[t_n,T]$. It is difficult to determine a good window size a *priori*, and the two programs do not use the same strategy for picking these windows. In RELAX2, an adaptive windowing algorithm is used, as the interaction between window size and converge speed is reasonably well understood for WR. However, in the case of WRN, a good window size tends to be a function of the equation nonlinearities rather than the differential equation dynamics. Since we did not have a good idea on how to adapt window sizes in this case, the SPLAX program uses relatively small window sizes compared to RELAX2.

The SPLAX program does not use the same trapezoidal discretization method, described in Section 2, as used in SPLICE3.1 and RELAX2. This is because the iterative timestep refinement technique used in SPLAX can use very large timesteps for early WRN iterations, and if the trapezoidal method were used, it would produce spurious oscillatory solutions(see [NAG75]). Although the timestep refinement would eventually remove the spurious oscillation, it might cost extra relaxation iterations.

To avoid this problem, the more stable second-order backward-difference method is used to solve the time varying linear iteration equation given in Algorithm 2. Specifically,

$$\begin{aligned}
&\alpha_1[q_i(v^{k+1,i}(t+h_f), u(t+h_f)) + \frac{\partial q_i(v^{k+1,i}(t+h_f), u(t+h_f))}{\partial v_i}(v_i^{k+1}(t+h_f) - v_i^k(t+h_f))] \\
&\quad +\alpha_2[q_i(v^{k+1,i}(t), u(t)) + \frac{\partial q_i(v^{k+1,i}(t), u(t))}{\partial v_i}(v_i^{k+p1}(t) - v_i^k(t))] \\
&\quad +\alpha_3[q_i(v^{k+1,i}(t+h_b), u(t+h_b)) + \frac{\partial q_i(v^{k+1,i}(t+h_b), u(t+h_b))}{\partial v_i}(v_i^{k+1}(t+h_b) - v_i^k(t+h_b))] \\
&\quad -[f_i(v^{k+1,i}(t+h_f), u(t+h_f)) + \frac{\partial f_i(v^{k+1,i}(t+h_f), u(t+h_f))}{\partial v_i}(v_i^{k+1}(t+h_f) - v_i^k(t+h_f))] = 0
\end{aligned}$$

where $\alpha_1, \alpha_2, \alpha_3$ are functions of the forward and backward timesteps, h_f and h_b (for the exact formulation, see [GEA74]). This would seem to put the SPLAX program at a slight disadvantage, as the second-order backward difference method does have larger truncation error than the trapezoidal method, and would normally require more timesteps for comparable accuracy. This does not seem to be the case however, perhaps because the backward-difference method is being used in combination with the a *posteriori* LTE control described above.

5.3 Simulation Results

In Table 1, the classical direct, ITA, standard WR, and WRN methods are compared using a number of example circuits. The examples are: a critical path from a microprocessor control circuit, *Microc*, the logic for a successive approximation register, *Scdac*, a dynamic memory cross-section, *Dram*, a static memory cross-section, *Sram*, and a digital filter, *Digfi*. In *Microc*, the waveforms exhibit multirate behavior mostly in the form of latency, and the trade-offs in WR and ITA balance to produce similar run times, but WRN is faster than either method since it exploits multirate behavior completely using relatively inexpensive iterations. The circuit is too small for relaxation methods to be of significant benefit though, and the classical direct method performs best. The circuits *Scdac* and *Dram exhibit* latency and are coupled, and therefore ITA is faster than WR, but WRN with its combined benefits is again faster than either. The WR algorithm is fastest for the circuit *Digfi*, because the partitioner breaks the circuit into completely unidirectional blocks, and therefore WR converges in one iteration. Note that for *Digfi*, WRN is 6 times slower than WR, and WRN proved to be even slower than direct methods! In *Sram*, WR is again the fastest, because although there is some localized coupling due to the gate-source and gate-drain capacitance (which are not included in the *Digfi* example), *Sram* is essentially unidirectional.

The above results imply that WRN is not very effective when used to simulate idealized MOS circuits which ignore gate-source and gate-drain capacitances. To provide stronger evidence of this behavior, *Sram* was simulated a number of times with a range of gate-source and gate-drain capacitances, as controlled by the thin-oxide thickness, *tox*, values. As Table 2 demonstrates, as the value of *tox* increases, and therefore the gate-source and gate-drain capacitance decreases, the ratio of the runtime of WRN to WR also increases.

An area of future work suggested by these results is to consider an algorithm which combines both the standard WR and WRN methods in one simulator. The choice of which method to use for a given circuit would then be based on the unidirectionality and linearity characteristics of the subcircuits that comprise the circuit. The portions of the circuit that contain elements with highly nonlinear device characteristics or exhibit predominantly unidirectional signal flow could be solved using WR while the remaining portions that feature moderate coupling or weakly nonlinear device characteristics could be solved using WRN. This modified WR method is expected to be even more effective as it exploits the advantages of standard WR and WRN. However, such an approach will require a method to automatically select the appropriate algorithm for each portion of the circuit and this will likely be the key research activity for this composite algorithm.

TABLE 1 - Direct vs SPLICE3 vs RELAX2 vs SPLAX

name	Nodes	Direct	ITA	WR	WRN
Microc	56	31.3	47.1	47.7	41.8
Scdac	150	290.1	302.1	355.8	278.5
Dram	300	650.2	582.3	861.7	535.9
Sram	129	333.9	238.7	178.8	331.3
Digfi	378	641.1	323.8	167.0	749.5

TABLE 2 - Runtime Ratios of SPLAX to RELAX2 as tox varies

tox	125*A*	250*A*	500*A*	1000*A*	∞
Splax/Relax2	1.7	1.85	2.3	2.4	8.9

6 Conclusions and Acknowledgements

A new circuit simulation algorithm, based on combining waveform-relaxation-Newton with an iterative step-size refinement scheme has been described and a new convergence proof for the WN has been presented. It has been shown experimentally that the new algorithm is effective for simulating MOS digital circuits, and compares well with with other relaxation algorithms when simulating moderately coupled, multirate circuits. It is not as well-suited to the simulation of unidirectional circuits, for which the standard WR is more appropriate, and has difficulty on certain highly-nonlinear problems. However, the WRN algorithm has certain features that can be exploited on parallel processors and this aspect provides an additional advantage over other methods.

We would like to thank Ken Kundert, Prof. O. Nevanlinna, F. Odeh, Albert Ruehli, and Don Webber for the many valuable discussions about waveform methods, and Gih-Guang Hung for generating the results.

This work was supported by the Defense Advanced Research Projects Agency contract NESC-N39 and N00014-87-K-825, Hewlett-Packard, and NSERC of Canada, and grants from the IBM company, Philips, Hewlett-Packard, Texas Instruments and MICRO.

References

[BOK83] W.M.G. van Bokhoven, "An Activity Controlled Modified Waveform Relaxation Method," *Proc. Int. Symp. on Circuits and Systems*, Newport Beach, California, May 1983.

[CHA75] B.R. Chawla, H.K. Gummel, and P. Kozak, "MOTIS - an MOS Timing Simulator," *IEEE Trans. on Circuits and Systems*, Vol. CAS-22, December 1975, pp. 901-909.

[CHE84] C. F. Chen and P. Subramanyam, "The Second Generation MOTIS Timing Simulator - An Efficient and Accurate Approach for General MOS Circuits" *Proc. Int. Symp. on Circuits and Systems*, Montreal, Canada, May 1984.

[CHU75] L. Chua and P. Lin, *Computer-Aided Analysis of Electronic Circuits: Algorithms and Computational Techniques*, Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1975.

[DEF84] P. Defebve, J. Beetem, W. Donath, H.Y. Hsieh, F. Odeh, A.E. Ruehli, P.K. Wolff, Sr., and J. White, "A Large-Scale Mosfet Circuit Analyser Based on Waveform Relaxation," *Proc. Int. Conf. on Computer Design*, Rye, New York, October 1984.

[DES69] C. A. Desoer and E. S. Kuh, *Basic Circuit Theory*, McGraw Hill, New York, 1969.

[DUM86] D. Dumlugol, P. Odent, J. Cockx, H. De Man, "The Segmented Waveform Relaxation Method for Mixed-Mode Switch Electrical Simulation of Digital MOS VLSI Circuits and its Hardware Acceleration on Parallel Computers," *Proc. Int. Conf. on Computer-Aided Design*, Santa Clara, California, September 1986.

[GEA74] C. William Gear, *Numerical Initial Value Problems for Ordinary Differential Equations*, Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1974.

[GEA80] C. William Gear, "Automatic Multirate Methods for Ordinary Differential Equations," *Information Processing 80*, North-Holland Pub. Co., Amsterdam, 1980.

[GUA83] M. Guarini and O. A. Palusinski, "Integration of Partitioned Dynamical Systems using Waveform Relaxation and Modified Functional Linearization," *Summer Computer Simulation Proceedings*, Vancouver, Canada, July 1983.

[HO75] C. W. Ho, A. E. Ruehli, and P. A. Brennan, "The Modified Nodal Approach to Network Analysis," *IEEE Trans. on Circuits and Systems*, Vol. CAS-22, June 1975, pp. 504-509.

[KAN64] Kantorovich, L. V. and G. P. Akilov, *Functional Analysis in Normed Spaces*. Pergammon, Oxford, 1964.

[KUN86] K. S. Kundert, "Sparse Matrix Techniques," *Circuit Analysis, Simulation and Design. Vol. 1.*, edited by A. E. Ruehli, North-Holland, 1986.

[LEL82] E. Lelarasmee, A. E. Ruehli, and A. L. Sangiovanni-Vincentelli, "The Waveform Relaxation Method for Time Domain Analysis of Large Scale Integrated Circuits," *IEEE Trans. on CAD of IC and Systems*, Vol. 1, No. 3, July 1982, pp. 131-145.

[NAG75] L.W. Nagel, "SPICE2: A computer program to simulate semiconductor circuits," Electronics Research Laboratory *Report No. ERL-M520*, University of California, Berkeley, May 1975.

[NEV86] O. Nevanlinna and F. Odeh, "Remarks on the Convergence of Waveform Relaxation Method" *Report MAT-A237*, Institute of Mathematics, Helsinki University of Technology, Finland, 1986.

[NEV87] O. Nevanlinna, Private Notes, 1987.

[NEW83] A.R. Newton and A. L. Sangiovanni-Vincentelli, "Relaxation-Based Circuit Simulation," *IEEE Trans. on ED*, Vol. ED-30, No. 9, Sept. 1983, pp. 1184-1207. Also *SIAM J. of Scientific and Stat. Computing*, Vol. 4, No. 3, September 1983, also *IEEE Trans. on CAD of IC and Systems*, Vol. CAD-3, No. 4, October 1984, pp. 308-330.

[ORT70] J. M. Ortega and W.C Rheinbolt, *Iterative Solution of Nonlinear Equations in Several Variables*, Academic Press, New York, 1970.

[SAL83] R. A. Saleh, J. E. Kleckner and A. R. Newton, "Iterated Timing Analysis and SPLICE1," *Proc. Int. Conf. on Computer-Aided Design*, Santa Clara, California, September 1983.

[SAL87A] R. Saleh and A. R. Newton, "An Event-Driven Relaxation-Based Multirate Integration Scheme for Circuit Simulation," *Proc. Int. Symp. on Circuits and Systems*, Philadelphia, Pennsylvania, May 1987.

[SAL87B] R. Saleh, D. Webber, E. Xia, A. Sangiovanni-Vincentelli, "Parallel Waveform Newton Algorithms for Circuit Simulation" *Proc. Int. Conf. Comp. Design* Rye, New York, October 87.

[STO80] J. Stoer and R. Bulirsch, *Introduction to Numerical Analysis* Springer-Verlag, New York, 1980.

[VAR62] R. S. Varga, *Matrix Iterative Analysis*, Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1962.

[WEE73] W. T. Weeks, A. J. Jimenez, G. W. Mahoney, D. Mehta, H. Qassemzadeh, and T. R. Scott, "Algorithms for ASTAP - A Network Analysis Program," *IEEE Trans. on Circuit Theory*, Vol. CT-20, No. 6, November 1973, pp. 628-634.

[WHI86] J. White and A. S. Vincentelli, *Relaxation Techniques for the Simulation of VLSI Circuits* Kluwer Academic Publishers, Norwell, Massachusetts, 1986.

A Proof of Theorem 3

The proof of Theorem 3 is somewhat complicated in detail. To avoid an unnecessarily lengthy presentation, but still convey where the necessary conditions stem from, several not quite obvious lemmas will presented without proof.

The following norm on the space of functions that map $[0,T] \rightarrow \mathbb{R}^n$ will be used throughout the proof.

Definition 1 *The β norm on a function $v : [0,T] \rightarrow \mathbb{R}^n$ is defined as $max_{\tau \in [0,T]} e^{-B\tau} \|v(t)\|^{\mathbb{R}^n}$ where B is some positive number. The β norm is denoted by $\|v\|_B$.*

Note that the β norm is really a continuous family of norms, one for each positive number B. On the space of continuous differentiable functions that map $[0,T] \rightarrow \mathbb{R}^n$, the β norm is equivalent to the norm used in the WN derivation.

Lemma 1 *If $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$ has a derivative $J_f : \mathbb{R}^n \rightarrow \in \mathbb{R}^{n \times n}$, which is Lipschitz continuous with some Lipschitz constant l_f, then*

$$\|f(y) - f(x) - J_f(x)(x-y)\| < l_f \|x-y\|^2 \tag{22}$$

Lemma 1 can be thought of as a remainder theorem for the Taylor series, the detailed proof can be found in [STO].

Lemma 2 *Given $F(v)$ as in (12) and $J_F(v)$ as in (16), it follows that*

$$\| \int_0^t F(x)(\tau) - F(y)(\tau) - J_F(y)(x-y)(\tau)d\tau \|_B < (l_{\dot{q}} + \frac{Tl_{\dot{g}}}{B})\|x-y\|_B^2 \tag{23}$$

for any x, y in the space of continuous differentiable functions that map $[0,T] \rightarrow I\!R^n$, where $l_{\dot{q}}$ is the Lipschitz constant for $\frac{\partial q}{\partial v}$ and $l_{\dot{g}}$ is the Lipschitz constant for $\frac{\partial g}{\partial v}$.

Lemma 2 can be derived by exploiting properties of the β norm[LEL82], and applying Lemma 1.

Lemma 3 *Let v^k, v^{k+1} be two iterates generated by (17), and $F(v)$ be as given in (12), then*

$$\|v^{k+1} - v^k\|_B < l_{\dot{q}^{-1}}(1 + \frac{Tl_{\dot{g}}}{B})\| \int_0^t F(v^k)(\tau)d\tau\|_B, \tag{24}$$

where $l_{\dot{q}^{-1}}$ is the bound on $\left(\frac{\partial q}{\partial v}\right)^{-1}$.

Lemma 3 can also be shown using β norm properties.

To prove the theorem, let v^k, and v^{k+1} be the k and $k+1^{th}$ WN iterates. Then

$$\| \int_0^t F(v^{k+1})(\tau) - F(v^k)(\tau) - J_F(v^k)(v^{k+1} - v^k)(\tau)d\tau\|_B < (l_{\dot{q}} + \frac{Tl_{\dot{g}}}{B})\|v^{k+1} - v^k\|_B^2 \tag{25}$$

by Lemma 1. By definition of the WN algorithm,

$$F(v^k)(\tau) - J_F(v^k)(v^{k+1} - v^k)(\tau) = 0, \tag{26}$$

and therefore (25) can be reduced to

$$\| \int_0^t F(v^{k+1})(\tau)d\tau\| < (l_{\dot{q}} + \frac{Tl_{\dot{g}}}{B})\|v^{k+1} - v^k\|_B^2. \tag{27}$$

By Lemma 2,

$$\frac{1}{l_{\dot{q}^{-1}}(1 + \frac{Tl_{\dot{g}}}{B})}\|v^{k+2} - v^{k+1}\|_B < \| \int_0^t F(v^k)(\tau)d\tau\|_B. \tag{28}$$

Substituting into equation (27) leads to

$$\|v^{k+2} - v^{k+1}\|_B < (l_{\dot{q}^{-1}}(1 + \frac{Tl_{\dot{g}}}{B}))(l_{\dot{q}} + \frac{Tl_{\dot{g}}}{B})\|v^{k+1} - v^k\|_B^2. \tag{29}$$

The theorem follows by noticing that the β norm of $\|v^{k+1} - v^k\|_B$ can be made as small as desired by increasing B, because $v^{k+1}(0) - v^k(0) = 0$. The result is that by picking a B large enough, (29) can be written as

$$\|v^{k+2} - v^{k+1}\|_B < \gamma\|v^{k+1} - v^k\|_B \tag{30}$$

where $\gamma < 1$ and is a function of B, l_{i-1}, l_i, l_j, and T. It therefore follows that v^k is a cauchy sequence and therefore converges ■