Accelerated Waveform Relaxation Techniques for the Parallel Transient Simulation of Semiconductor Devices

by

Mark W. Reichelt

S.B. (EE & CS), Massachusetts Institute of Technology (1987)
S.M. (EE & CS), Massachusetts Institute of Technology (1987)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

Massachusetts Institute of Technology

June 1993

Signature of Author_____
Department of Electrical Engineering and Computer Science
May 14, 1993

Certified by_____
Jacob K. White
Thesis Supervisor

Accepted by_____
Campbell L. Searle
Chair, Departmental Committee on Graduate Students

Accelerated Waveform Relaxation Techniques for the Parallel Transient Simulation of Semiconductor Devices

by

Mark W. Reichelt

Submitted to the Department of Electrical Engineering and Computer Science
on May 14, 1993, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

This thesis contains a description of a novel generalized SOR algorithm for accelerating the convergence of the dynamic iteration method known as waveform relaxation. The new waveform convolution SOR algorithm is presented, along with a theorem for determining the optimal convolution SOR parameter. Both analytic and experimental results are given to demonstrate that the convergence of the waveform convolution SOR algorithm is substantially faster than that of the more obvious ordinary waveform SOR algorithm. To demonstrate the general applicability of this new method, it is used to solve the differential-algebraic system generated by spatial discretization of the time-dependent semiconductor device equations.

The accelerated waveform relaxation algorithm is compared to pointwise direct and iterative methods for the transient simulation of semiconductor devices on both serial and parallel machines. In particular, experimental results are presented for simulations on a small cluster of workstations running the Parallel Virtual Machine (PVM) software, as well as for simulations on a 32-processor Intel iPSC/860. The results show that the accelerated waveform method is competitive with standard pointwise methods on serial machines, and is significantly faster on commonly available loosely-coupled MIMD machines. The parallel accelerated waveform method achieves nearly linear speed-up on the 32 processor Intel iPSC/860 hypercube. The strong implication of the results is that, as MIMD machines become more prevalent, accelerated waveform methods may gain in importance for all areas of simulation requiring the solution of initial value problems.

Thesis Supervisor: Jacob K. White

Associate Professor

To Kim, with thanks and all my love

Acknowledgments

First, I thank my mother and father, brother, extended family and in-laws for unending support and encouragement.

I thank my advisor, Prof. Jacob White, who made this work possible with his support and guidance. Thanks also to Prof. Jonathan Allen for his support, and to Prof. Dimitri Antoniadis for his advice and interest.

Special thanks to my friends on the 8th floor, including but not limited to, Don Baltus, Bob Armstrong, Andrew Lumsdaine, Abe Elfadel, Steve McCormick, Matt Kamon, Joel Phillips, Andrew Grumet, Ricardo Telichevesky, Keith Nabors, Miguel Silveira, Mark Seidel, Khalid Rahmat, Ig McQuirk, Chris Umminger, Amelia Shen, Jose Monteiro and Denny Freeman. Don was a perfect office-mate; he, Bob and Abe were always there for technical and philosophical discussions. I thank Prof. Lumsdaine for his collaboration on the pWORDS project, and for encouraging words when times got hectic.

I would also like to thank the many people I have encountered along the way for their valuable suggestions and their friendship. These people include: Professors Nick Trefethen, Don Rose, Paul Lanzcron, and Alar Toomre; my friends at Intel, especially Don Scharfetter, Tim Thurgate and Peter Saviz; and my friends at Bell Labs, especially Wayne Wolf, Wolfgang Fichtner, Mark Pinto, Enrico SanGiorgi, Kurt Keutzer, Al Dunlop, Ted Kowalski, and Mike McFarland.

Above all, I give thanks to my wife Kimberly, who has helped me through it all and more. I cannot thank her enough, though I will continue to try.

This work was supported by a grant from IBM, the Defense Advanced Research Projects Agency contracts N00014-87-K-825 and MIP-88-14612, the National Science Foundation.

Contents

List of Figures

List of Tables

Tho' much is taken, much abides; and tho'
We are not now that strength which in old days
Moved earth and heaven, that which we are, we are,—
One equal temper of heroic hearts,
Made weak by time and fate, but strong in will
To strive, to seek, to find, and not to yield.

— ALFRED TENNYSON, Ulysses (1833)

1

Introduction

1.1 Initial Value Problems on Parallel Machines

Simulation tools for the analysis of complex systems have become indispensable for engineering design and testing. As in almost any enterprise, success in the art of simulation depends on using the right tool for the right job. In today's ever evolving computing environment, this often means combining a specific numerical algorithm with a specific type of computer architecture.

At the heart of many interesting and important simulation problems, such as the transient simulation of semiconductor devices, there lies an initial value problem, a system of differential equations in time with a known initial condition. This thesis will show that a particularly efficient way to solve the initial value problem arising from device simulation is to use accelerated *waveform relaxation* algorithms on coarse-grain, loosely-coupled parallel machines.

The primary theoretical contribution of this thesis is the introduction and development of *convolution SOR*, a novel technique for accelerating the convergence of waveform relaxation. Using a suite of device transient simulation examples, the performance of the waveform relaxation algorithm, accelerated with convolution SOR, is compared to that of standard direct and iterative methods on two commonly-available parallel machines, a small cluster of workstations running the Parallel Virtual Machine software, and a 32-processor Intel iPSC/860.

The results show that accelerated waveform methods are competitive with standard pointwise methods on serial machines, and that the accelerated waveform method is significantly faster on the parallel machines. In particular, the parallel accelerated waveform method achieves nearly linear speed-up on the 32 processor Intel iPSC/860 hypercube,

whereas experiments with parallel versions of standard pointwise methods do not exhibit any parallel speed-up. The strong implication of these results is that, as MIMD (Multiple Instruction Multiple Data) parallel machines become more prevalent, accelerated waveform methods will gain in importance for all areas of simulation requiring the solution of initial value problems.

1.2 Device Transient Simulation

An initial value problem (IVP) which can be used to simulate the transient behavior of semiconductor devices is derived from simplified device physics. Charge transport within a semiconductor device is assumed to be governed by the Poisson equation, and the electron and hole continuity equations:

$$\frac{kT}{q}\nabla\cdot(\epsilon\nabla u)+q\,(p-n+N_D-N_A) = 0 \tag{1.1}$$

$$\nabla\cdot\boldsymbol{J}_n-q\left(\frac{\partial n}{\partial t}+R\right) = 0 \tag{1.2}$$

$$\nabla\cdot\boldsymbol{J}_p+q\left(\frac{\partial p}{\partial t}+R\right) = 0 \tag{1.3}$$

where u is the normalized electrostatic potential in thermal volts, n and p are the electron and hole concentrations, $\boldsymbol{J}_n$ and $\boldsymbol{J}_p$ are the electron and hole current densities, N_D and N_A are the donor and acceptor concentrations, R is the net generation and recombination rate, q is the magnitude of the charge of an electron, and ϵ is the spatially-dependent dielectric permittivity [2, 41].

The current densities $\boldsymbol{J}_n$ and $\boldsymbol{J}_p$ are given by the drift-diffusion approximations:

$$\boldsymbol{J}_n = -q\mu_n n\,\nabla\left(\frac{kT}{q}\,u\right)+qD_n\nabla n = -kT\mu_n n\nabla u+qD_n\nabla n \tag{1.4}$$

$$\boldsymbol{J}_p = -q\mu_p p\,\nabla\left(\frac{kT}{q}\,u\right)-qD_p\nabla p = -kT\mu_p p\nabla u-qD_p\nabla p \tag{1.5}$$

where μ_n and μ_p are the electron and hole mobilities, and D_n and D_p are the diffusion coefficients. The diffusion constants D_n and D_p are related to the mobilities by the Einstein relations

$$D_n=\frac{kT}{q}\,\mu_n \quad\text{and}\quad D_p=\frac{kT}{q}\,\mu_p.$$

The mobilities μ_n and μ_p are used to model many physical mechanisms. One standard approach is to model the mobilities as nonlinear functions of the electric field E, i.e.

$$\mu_n=\mu_{n_0}\left[1+\left(\frac{\mu_{n_0}E}{v_{sat}}\right)^{\beta}\right]^{-(1/\beta)}$$

where v_{sat} and β are constants and μ_{n_0} is a doping-dependent mobility [26]. The drift-diffusion approximations (1.4) and (1.5) are typically used to eliminate the current densities $\boldsymbol{J}_n$ and $\boldsymbol{J}_p$ from the continuity equations (1.7) and (1.8),

$$\frac{kT}{q}\nabla\cdot(\epsilon\nabla u)+q\,(p-n+N_D-N_A) = 0 \tag{1.6}$$

$$\nabla\cdot(-kT\mu_n n\nabla u+qD_n\nabla n)-q\left(\frac{\partial n}{\partial t}+R\right) = 0 \tag{1.7}$$

$$\nabla\cdot(-kT\mu_p p\nabla u-qD_p\nabla p)+q\left(\frac{\partial p}{\partial t}+R\right) = 0 \tag{1.8}$$

leaving a differential-algebraic system of three equations (1.6)–(1.8) in three unknowns, u, n, and p.

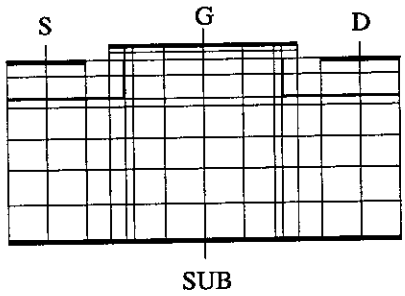

FIGURE 1-1: Example of a rectangular discretization mesh covering a two-dimensional slice of a MOSFET device, indicating the gate, source, drain and substrate contacts.

Given a rectangular mesh covering a two-dimensional slice of a MOSFET, such as that shown in Figure 1-1, a common approach to spatially discretizing the device equation system (1.6)–(1.8) is to use a finite-difference formula to discretize the Poisson equation, and a exponentially-fit finite-difference formula to discretize the continuity equations (the Scharfetter-Gummel method [40]). On an N-node mesh, this spatial discretization yields a sparsely-coupled nonlinear initial value problem consisting of $3N$ equations in $3N$ unknowns, denoted by

$$\boldsymbol{F}_1(\boldsymbol{u}(t),\boldsymbol{n}(t),\boldsymbol{p}(t),t) = 0 \tag{1.9}$$

$$\tfrac{d}{dt}\boldsymbol{n}+\boldsymbol{F}_2(\boldsymbol{u}(t),\boldsymbol{n}(t),\boldsymbol{p}(t),t) = 0 \tag{1.10}$$

$$\tfrac{d}{dt}\boldsymbol{p}+\boldsymbol{F}_3(\boldsymbol{u}(t),\boldsymbol{n}(t),\boldsymbol{p}(t),t) = 0 \tag{1.11}$$

$$\boldsymbol{u}(0) = \boldsymbol{u}_0$$

$$\boldsymbol{n}(0) = \boldsymbol{n}_0$$

$$\boldsymbol{p}(0) = \boldsymbol{p}_0$$

where $t \in [0,T]$, and $\boldsymbol{u}(t), \boldsymbol{n}(t), \boldsymbol{p}(t) \in \mathbb{R}^N$ are vectors of normalized potential, electron concentration, and hole concentration. Note that because the Poisson equation is an algebraic constraint with no time derivative term, the device equation system is a differential-algebraic (DAE) system.

In equations (1.9)–(1.11), $\boldsymbol{F}_1, \boldsymbol{F}_2, \boldsymbol{F}_3 : \mathbb{R}^{3N} \to \mathbb{R}^N$ can be specified component-wise as

$$F_{1_i}(u_i, n_i, p_i, u_j) = \frac{kT}{q} \sum_j \left\{ \frac{d_{ij}\,\epsilon_{ij}}{L_{ij}} \left[u_i - u_j \right] \right\} - qA_i\,(p_i - n_i + N_{D_i} - N_{A_i}) \quad (1.12)$$

$$F_{2_i}(u_i, n_i, u_j, n_j) = \frac{kT}{q} \sum_j \left\{ \frac{d_{ij}\,\mu_{n_{ij}}}{L_{ij}} \left[n_i B(u_i - u_j) - n_j B(u_j - u_i) \right] \right\} + A_i\,R_i \quad (1.13)$$

$$F_{3_i}(u_i, p_i, u_j, p_j) = \frac{kT}{q} \sum_j \left\{ \frac{d_{ij}\,\mu_{p_{ij}}}{L_{ij}} \left[p_i B(u_j - u_i) - p_j B(u_i - u_j) \right] \right\} + A_i\,R_i. \quad (1.14)$$

The summations are taken over the silicon nodes j adjacent to node i. As shown in Figure 1-2, for each node j adjacent to node i, L_{ij} is the distance from node i to node j, d_{ij} is the length of the side of the Voronoi box that encloses node i and bisects the edge between nodes i and j, and A_i is the area of the Voronoi box. Similarly, the quantities ϵ_{ij}, $\mu_{n_{ij}}$ and $\mu_{p_{ij}}$ are the dielectric permittivity, electron and hole mobility, respectively, on the edge between nodes i and j. The Bernoulli function, $B(x) = x/(e^x - 1)$, is used to exponentially fit potential variation to electron and hole concentration variations, and effectively upwinds the current equations.

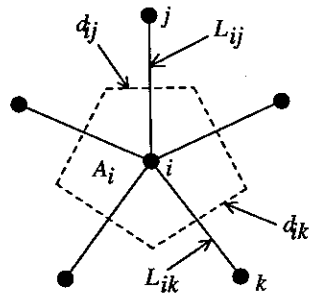

FIGURE 1-2: Illustration of a mesh node i, the area A_i of its Voronoi box, and the lengths d_{ij} and L_{ij}.

The standard approach used to solve the device differential-algebraic equation system (1.9)–(1.11) is to discretize the system in time with a low-order implicit integration method such as the second-order backward difference formula (BDF) or a hybrid trapezoidal/BDF method [2]. For an N-node mesh, the resulting sequence of nonlinear algebraic systems in $3N$ unknowns is typically solved with some variant of Newton's method and/or relaxation [2, 23]. This approach can be disadvantageous for a parallel implementation, especially for MIMD parallel computers having a high communication

latency, since the processors will have to synchronize repeatedly (at every iteration) for each timestep. The following chapters will show that a more effective approach to solving (1.9)-(1.11) with a parallel computer is to decompose the device equation system into subsystems before time discretization, with a waveform relaxation (WR) algorithm [18, 33, 22].

1.3 Overview

Chapter 2 is a review of numerical techniques relevant to device simulation. In Chapter 3, the waveform relaxation (WR) algorithm for device simulation is examined, and theorems are presented guaranteeing convergence of the method. Chapter 4 is a detailed description of the implementation of WR for devices used in the WORDS simulation program. The convolution successive overrelaxation (CSOR) acceleration technique is introduced and developed in Chapter 5. Parallel implementations of the simulation methods and parallel experimental results are given in Chapter 6. Finally, Chapter 7 summarizes the contributions of this thesis.

2

Review of Numerical Techniques

2.1 Introduction

In this chapter, some of the techniques for solving time-dependent nonlinear initial-value problems are briefly reviewed. Because there are hundreds of different techniques, and dozens of excellent texts on the subject (for example [7, 9, 11, 12, 13, 29, 28, 44, 48, 52]), the list of methods mentioned in this chapter is by no means exhaustive. Rather, the following is intended only as a review of some of the methods relevant to the transient simulation of semiconductor devices with standard and waveform methods.

2.2 Solution of Linear Systems

This section describes a few of the standard solution methods for solving linear systems of the form

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \tag{2.1}$$

where $\boldsymbol{A} \in \mathbb{R}^{n\times n}$, $\boldsymbol{b} \in \mathbb{R}^n$, and vector $\boldsymbol{x} \in \mathbb{R}^n$ is the unknown. Because of their importance and variety of application, there are many different numerical methods for solving linear systems. These methods lie at the heart of most nonlinear system solution methods. Though this is a very mature field, there is still active research.

2.2.1 Gaussian Elimination

Gaussian elimination is a process of $\boldsymbol{LU}$-factorization, in which the non-singular matrix $\boldsymbol{A}$ of (2.1) is factored into a product of a lower triangular matrix $\boldsymbol{L}$ and an upper triangular matrix $\boldsymbol{U}$. It is probably the most widely used of all numerical algorithms,

and accordingly, descriptions of the method and its variants can be found in most linear algebra texts [45, 12]. The solution $\boldsymbol{x}^* = \boldsymbol{A}^{-1}\boldsymbol{b}$ to (2.1) is computed by first solving the lower triangular matrix problem $\boldsymbol{L}\boldsymbol{y} = \boldsymbol{b}$ in a process known as *forward elimination* and then solving the upper triangular matrix problem $\boldsymbol{U}\boldsymbol{x} = \boldsymbol{y}$ in a process known as *backward substitution.*

As a direct method for solving (2.1), elimination produces the solution $\boldsymbol{x}^*$ in a finite number of steps for *any* non-singular matrix $\boldsymbol{A}$ (assuming exact arithmetic). Though an extremely robust algorithm, the disadvantage of elimination is its computational and storage cost. In general, elimination requires $\mathcal{O}(n^3)$ operations and $\mathcal{O}(n^2)$ storage. For *sparse* problems, in which $\boldsymbol{A}$ contains many more zeros than non-zeros, the computational cost of elimination depends enormously on the sparsity pattern of the matrix, e.g. matrices associated with trees factor in $\mathcal{O}(n)$ operations. For sparse matrices arising from 2D finite-difference discretizations, the operation count of elimination is about $\mathcal{O}(n^{1.5})$, and for 3D discretizations, the operation count is about $\mathcal{O}(n^2)$ [10]. During the forward elimination and backward substitution phases, the matrix $\boldsymbol{A}$ and its $\boldsymbol{LU}$ factorization must be stored in memory, requiring up to $\mathcal{O}(n^2)$ storage. Finally, Gaussian elimination is difficult to parallelize efficiently. This, and the tradeoff between general reliability and cost has led to a proliferation of highly competitive iterative methods.

2.2.2 Gauss-Jacobi and Gauss-Seidel Relaxation

An iterative method for solving $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ begins with a guess solution $\boldsymbol{x}^0 \in \mathbb{R}^n$, and operates on this solution in some way to produce a sequence of solutions $\{\boldsymbol{x}^k\}$ that approach the exact solution $\boldsymbol{x}^* = \boldsymbol{A}^{-1}\boldsymbol{b}$ as $k \to \infty$. Typically, if $\boldsymbol{A}$ satisfies certain properties, then the convergence of $\boldsymbol{x}^k$ to $\boldsymbol{x}^*$ for any initial guess $\boldsymbol{x}^0$ can be guaranteed. Of course, the conditions on $\boldsymbol{A}$ depend on the iterative method used.

Two simple iterative methods are the Gauss-Jacobi (GJ) and Gauss-Seidel (GS) relaxation algorithms [48, 52, 28], which generate the new solution $\boldsymbol{x}^{k+1}$ by solving for element i of $\boldsymbol{x}^{k+1}$ with the equations

$$a_{ii}\, x_i^{k+1} = b_i - \sum_{i \neq j} a_{ij}\, x_j^k, \tag{2.2}$$

for GJ relaxation, or

$$a_{ii}\, x_i^{k+1} = b_i - \sum_{j=1}^{i-1} a_{ij}\, x_j^{k+1} - \sum_{j=i+1}^{n} a_{ij}\, x_j^k, \tag{2.3}$$

for GS relaxation.

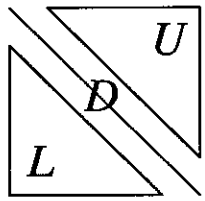

FIGURE 2-1: With the standard relaxation splitting, matrix $\boldsymbol{A}$ is structurally split into its lower triangular, upper triangular, and diagonal pieces such that $\boldsymbol{A} = \boldsymbol{D} - \boldsymbol{L} - \boldsymbol{U}$.

Let $\boldsymbol{L}, \boldsymbol{U}, \boldsymbol{D} \in \mathbb{R}^{n \times n}$ denote the strictly lower triangular, strictly upper triangular, and diagonal pieces of $\boldsymbol{A}$, such that $\boldsymbol{A} = \boldsymbol{D} - \boldsymbol{L} - \boldsymbol{U}$, as shown in Figure 2-1. This splitting should not be confused with the $\boldsymbol{A} = \boldsymbol{LU}$ of Gaussian elimination. With the relaxation splitting, the GJ and GS update equations (2.2) and (2.3), respectively, may be expressed succinctly in matrix form as

$$\boldsymbol{D}\boldsymbol{x}^{k+1} = \boldsymbol{b} + (\boldsymbol{L} + \boldsymbol{U})\boldsymbol{x}^k \tag{2.4}$$

$$(\boldsymbol{L} + \boldsymbol{D})\boldsymbol{x}^{k+1} = \boldsymbol{b} + \boldsymbol{U}\boldsymbol{x}^k. \tag{2.5}$$

Subtracting successive iterations and denoting $\Delta\boldsymbol{x}^{k+1} = \boldsymbol{x}^{k+1} - \boldsymbol{x}^k$ yields iteration equations

$$\Delta\boldsymbol{x}^{k+1} = \boldsymbol{H}\,\Delta\boldsymbol{x}^k \tag{2.6}$$

where the iteration matrices $\boldsymbol{H}_{GJ}, \boldsymbol{H}_{GS} \in \mathbb{R}^{n \times n}$ are given by

$$\boldsymbol{H}_{GJ} = \boldsymbol{D}^{-1}(\boldsymbol{L} + \boldsymbol{U}) \tag{2.7}$$

$$\boldsymbol{H}_{GS} = (\boldsymbol{D} + \boldsymbol{L})^{-1}\boldsymbol{U} \tag{2.8}$$

The advantage of representing the relaxation process with equation (2.6) is that the restrictions on $\boldsymbol{A}$ to obtain convergence may be concisely expressed in terms of the spectral radius $\rho(\boldsymbol{H})$, defined as

$$\rho(\boldsymbol{H}) \equiv \max_{1 \le i \le n} |\lambda_i| \tag{2.9}$$

where λ_i are the eigenvalues of $\boldsymbol{H}$ [48, 52, 28].

THEOREM 2.2.1. *If $\boldsymbol{D}^{-1}$ and $\boldsymbol{A}^{-1}$ exist, then the iterates $\boldsymbol{x}^k$ produced by iteration equation (2.6) converge to $\boldsymbol{x}^* = \boldsymbol{A}^{-1}\boldsymbol{b}$ for any initial guess $\boldsymbol{x}^0$ if and only if $\rho(\boldsymbol{H}) < 1$.*

For GJ relaxation, the requirement that $\rho(\boldsymbol{H}_{GJ}) < 1$ is satisfied by the common class of *irreducibly diagonally dominant* matrices. Such matrices arise naturally from finite difference discretizations of differential equations. Their significance is given by the following definitions and theorem.

DEFINITION 2.2.2. *A matrix* $\boldsymbol{A} \in \mathbb{C}^{n\times n}$ *is* irreducible *if there does not exist a permutation matrix* $\boldsymbol{P}$ *such that*

$$\boldsymbol{PAP}^{-1} = \begin{bmatrix} \boldsymbol{B}_{11} & \boldsymbol{B}_{12} \\ 0 & \boldsymbol{B}_{22} \end{bmatrix} \tag{2.10}$$

where $\boldsymbol{B}_{11}$ *and* $\boldsymbol{B}_{22}$ *are square matrices. Equivalently,* $\boldsymbol{A}$ *is irreducible if and only if for any two distinct indices* $1 \leq i, j \leq n$*, there is a sequence of nonzero elements of* $\boldsymbol{A}$ *of the form*

$$\{a_{i,k_1}, a_{k_1,k_2}, \ldots, a_{k_m,j}\} \tag{2.11}$$

DEFINITION 2.2.3. *A matrix* $\boldsymbol{A} \in \mathbb{C}^{n\times n}$ *is* diagonally dominant *if*

$$|a_{ii}| \geq \sum_{j\neq i} |a_{ij}| \quad \textit{for} \quad i = 1, \ldots, n \tag{2.12}$$

and strictly diagonally dominant *if strict inequality hold in (2.12) for all* i*. The matrix is* irreducibly diagonally dominant *if it is irreducible, diagonally dominant, and strict inequality holds in (2.12) for at least one* i*.*

THEOREM 2.2.4. *If* $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ *is either strictly or irreducibly diagonally dominant then* $\rho(\boldsymbol{H}_{GJ}) < 1$ *and* $\rho(\boldsymbol{H}_{GS}) < 1$*, where* $\boldsymbol{H}_{GJ}$ *and* $\boldsymbol{H}_{GS}$ *are given by (2.7)–(2.8).*

In an implementation on a serial machine, the primary difference between the GJ and GS relaxation equations (2.2) and (2.3) is that the GS equation immediately uses the new value x_i^{k+1} to compute all components x_j^{k+1} where $j > i$. Although this typically causes GS relaxation to converge more quickly than GJ, it also implies that the GS algorithm depends on the specific ordering of the unknowns. This dependence on serial ordering implies that the GS method is not as easily parallelized as GJ relaxation, and leads to the conclusion that on a parallel machine, GJ relaxation is faster [43].

For problems spatially discretized on rectangular meshes, one way to parallelize GS is to color the nodes red and black in an alternating or checkerboard fashion, as shown in Figure 2-2, such that red nodes have only black nodes as neighbors and the black nodes have only red nodes as neighbors. Then, with a simple finite-difference discretization, the variables at red nodes depend only on the variables at black nodes, and all of the red nodes may be solved simultaneously in parallel. Similarly, all of the black nodes may be solved simultaneously in parallel.

An additional advantage of using red/black ordering is that the resulting matrices belong to a common class known as *consistently ordered* matrices. As will be discussed

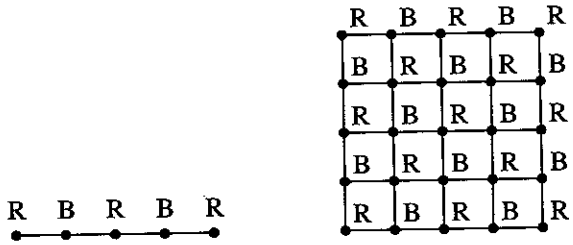

FIGURE 2-2: Illustration of red/black coloring for one and two dimensional meshes.

in the next section, it is possible to accelerate the convergence of GS relaxation on consistently ordered matrices with the successive overrelaxation (SOR) technique, without sacrificing the parallelizability of the red/black GS method. The result is that parallel red/black SOR can be significantly faster than the unaccelerated parallel GJ or GS relaxation algorithms.

2.2.3 Successive Overrelaxation

The convergence of GS relaxation may be accelerated with a technique known as successive overrelaxation (SOR) [48, 52]. To derive the SOR iteration equation, compute an intermediate value $\hat{x}_i^{k+1}$ by solving

$$a_{ii}\,\hat{x}_i^{k+1} = b_i - \sum_{j=1}^{i-1} a_{ij}\,x_i^{k+1} - \sum_{j=i+1}^{n} a_{ij}\,x_i^k, \tag{2.13}$$

similar to the GS equation (2.3), and then update x_i^k in the iteration direction by multiplication with an overrelaxation parameter $\omega \in \mathbb{R}$,

$$x_i^{k+1} \leftarrow x_i^k + \omega \cdot \left[\hat{x}_i^{k+1} - x_i^k\right]. \tag{2.14}$$

Combining equations (2.13) and (2.14) yields

$$a_{ii}\,x_i^{k+1} = (1-\omega)\,a_{ii}\,x_i^k + \omega\left[b_i - \sum_{j=1}^{i-1} a_{ij}x_i^{k+1} - \sum_{j=i+1}^{n} a_{ij}x_i^k\right]. \tag{2.15}$$

After subtracting successive waveform relaxation iterations, and using the relaxation splitting $\boldsymbol{A} = \boldsymbol{D} - \boldsymbol{L} - \boldsymbol{U}$, this leads to

$$(\boldsymbol{D} - \omega\boldsymbol{L})\,\Delta\boldsymbol{x}^{k+1} = [(1-\omega)\boldsymbol{D} + \omega\boldsymbol{U}]\,\Delta\boldsymbol{x}^k, \tag{2.16}$$

where $\Delta\boldsymbol{x}^{k+1} = \boldsymbol{x}^{k+1} - \boldsymbol{x}^k$. Therefore the SOR iteration equation may be written as

$$\Delta\boldsymbol{x}^{k+1} = \boldsymbol{H}_{SOR}\,\Delta\boldsymbol{x}^k \tag{2.17}$$

where $\boldsymbol{H}_{SOR} \in \mathbb{R}^{n\times n}$ is dependent on the choice of SOR parameter ω, and is given by

$$\boldsymbol{H}_{SOR} = (\boldsymbol{D} - \omega\boldsymbol{L})^{-1}[(1-\omega)\boldsymbol{D} + \omega\boldsymbol{U}]. \tag{2.18}$$

Note that when $\omega = 1$,

$$\boldsymbol{H}_{SOR} = \boldsymbol{H}_{GS}, \tag{2.19}$$

and the SOR method reduces to ordinary GS relaxation.

The rate of convergence of SOR is critically dependent on the selection of the SOR parameter ω. Indeed, incorrectly chosen values of ω may cause $\rho(\boldsymbol{H}_{SOR}) > 1$ so that SOR does not converge. One general restriction on ω is given by the following theorem, which implies that for any matrix, SOR will diverge unless $0 < \omega < 2$ if $\omega \in \mathbb{R}$.

THEOREM 2.2.5 (KAHAN). *If $\boldsymbol{A} \in \mathbb{C}^{n\times n}$ has nonzero diagonal elements, then*

$$\rho(\boldsymbol{H}_{SOR}) \geq |\omega - 1| \tag{2.20}$$

For a class of matrices known as *consistently ordered* matrices, it is possible to choose an optimal SOR parameter ω_{opt}, that minimizes the spectral radius of $\boldsymbol{H}_{SOR}$ over all possible ω. Such matrices naturally arise from finite difference discretizations of differential equations. In the following, we will use Young's definition of the class of consistently ordered matrices.

DEFINITION 2.2.6. *The $n \times n$ matrix A is consistently ordered if for some t there exist disjoint nonempty subsets $S_1, \ldots, S_t$ of $\{1, 2, \ldots, n\}$, with $\bigcup_{i=1}^{t} S_i = \{1, \ldots, n\}$, such that if $A_{ij} \neq 0$ with $i \neq j$ and S_k is the subset containing i, then $j \in S_{k+1}$ if $j > i$ and $j \in S_{k-1}$ if $j < i$.*

Note that consistent ordering is a purely structural property of a matrix. The importance of the class of consistently ordered matrices stems from three classic results [51, 48]:

LEMMA 2.2.7 (YOUNG,VARGA). *Let $\boldsymbol{A} = \boldsymbol{D} - \boldsymbol{L} - \boldsymbol{U}$ be a consistently ordered matrix, and let $\boldsymbol{H}_{GJ}$ denote the corresponding GJ iteration matrix (2.7). If μ_i is an eigenvalue $\boldsymbol{H}_{GJ}$ then $-\mu_i$ is also an eigenvalue of $\boldsymbol{H}_{GJ}$.*

THEOREM 2.2.8 (YOUNG,VARGA). *Let $\boldsymbol{A} = \boldsymbol{D} - \boldsymbol{L} - \boldsymbol{U}$ be a consistently ordered matrix, let $\boldsymbol{H}_{GJ}$ denote the corresponding GJ iteration matrix (2.7), and let $\boldsymbol{H}_{SOR}$ denote*

the SOR iteration matrix (2.18). Given an overrelaxation parameter $\omega \neq 0$, if λ is a nonzero eigenvalue of $\boldsymbol{H}_{SOR}$, then

$$(\lambda + \omega - 1)^2 = \omega^2 \mu^2 \lambda, \tag{2.21}$$

where μ is an eigenvalue of $\boldsymbol{H}_{GJ}$. Conversely, if μ is an eigenvalue of $\boldsymbol{H}_{GJ}$ and if λ satisfies (2.21), then λ is an eigenvalue of $\boldsymbol{H}_{SOR}$.

COROLLARY 2.2.9. *If $\boldsymbol{A} = \boldsymbol{D} - \boldsymbol{L} - \boldsymbol{U}$ is a consistently ordered matrix, then the eigenvalues of $\boldsymbol{H}_{GS}$ are the squares of the eigenvalues of $\boldsymbol{H}_{GJ}$.*

THEOREM 2.2.10 (YOUNG,VARGA). *If $\boldsymbol{A} = \boldsymbol{D} - \boldsymbol{L} - \boldsymbol{U}$ is a consistently ordered matrix, if the eigenvalues μ_i of the corresponding GJ iteration matrix $\boldsymbol{H}_{GJ}$ lie on the real-axis line segment $[-\mu_1, \mu_1]$, and if the spectral radius $\rho(\boldsymbol{H}_{GJ}) = |\mu_1| < 1$, then the spectral radius of the SOR iteration matrix $\boldsymbol{H}_{SOR}(\omega)$ is minimized by an overrelaxation parameter $\omega \in [1, 2]$ given by*

$$\omega_{opt} = \frac{2}{1 + \sqrt{1 - \mu_1^2}} \tag{2.22}$$

where $\sqrt{\cdot}$ denotes the positive root.

The SOR method, using the parameter ω_{opt} computed from (2.22) is surprisingly robust and is remarkably successful in practice, even when the conditions of Theorem 2.2.10 aren't precisely satisfied (i.e. the matrix $\boldsymbol{A}$ isn't consistently ordered, or all of the spectrum of $\boldsymbol{H}_{GJ}$ does not lie on the real line). Because of this, the SOR algorithm is a remarkably simple and effective iterative method. In general, for an elliptic problem such as the Poisson equation, discretized on a one-dimensional mesh of N evenly-space nodes or on a two-dimensional $N \times N$ rectangular mesh, the asymptotic convergence rate of SOR is approximately proportional to $1 - 1/N$. For large N, this can be significantly faster than the asymptotic convergence rate of Gauss-Jacobi or Gauss-Seidel relaxation, approximately proportional to $1 - 1/N^2$ [13].

As mentioned in the previous section, a red/black ordered problem leads to consistently ordered matrices, so that the optimum SOR parameter ω_{opt} may be computed for red/black SOR . The implication of this is that on a parallel machine, the red/black SOR method can be substantially faster than the unaccelerated GJ or GS relaxation algorithms.

2.2.4 Block Relaxation

An additional way to accelerate a relaxation method is to solve larger pieces of the problem directly, with block relaxation [48]. In other words, rather than decomposing

the system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ into n scalar algebraic equations, it may be possible to obtain faster relaxation convergence by decomposing the system into $m < n$ smaller systems of equations, where

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{A}_{11} & \boldsymbol{A}_{12} & \cdots & \boldsymbol{A}_{1m} \\ \boldsymbol{A}_{21} & \boldsymbol{A}_{22} & \cdots & \boldsymbol{A}_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{A}_{m1} & \boldsymbol{A}_{m2} & \cdots & \boldsymbol{A}_{mm} \end{bmatrix}.$$

Then, for example, block GS relaxation is performed by iterating over the m blocks and repeatedly solving the small matrix equation

$$\boldsymbol{A}_{ii}\,\boldsymbol{x}_i^{k+1} = \boldsymbol{b}_i - \sum_{j=1}^{i-1} \boldsymbol{A}_{ij}\,\boldsymbol{x}_j^{k+1} - \sum_{j=i+1}^{n} \boldsymbol{A}_{ij}\,\boldsymbol{x}_j^{k}, \tag{2.23}$$

where $\boldsymbol{x}_i$ denotes the piece of the vector $\boldsymbol{x}$ corresponding to block row i. Provided that the block relaxation converges quickly enough, this still provides a computational savings over direct methods because the sub-systems are smaller.

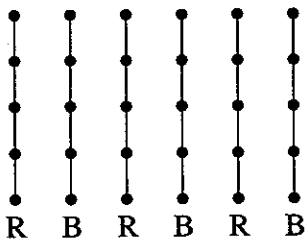

FIGURE 2-3: Illustration of two possible red/black colorings for block relaxation.

The SOR theorems of the previous section carry over for the block SOR method [48]. In particular, note that Definition 2.2.6 can be used to define *block consistently ordered* matrices. Furthermore, with properly chosen blocks, such as those shown in Figure 2-3, block GS and block SOR algorithms can use red/black ordering and maintain parallelizability. In practice, the block Gauss-Jacobi, block Gauss-Seidel and block SOR methods can converge substantially faster than ordinary relaxation, particularly if the difficult, tightly-coupled parts of the problem are contained within single blocks.

2.3 Solution of Nonlinear Systems

This section is a review of standard methods for solving systems of nonlinear equations

$$\boldsymbol{F}(\boldsymbol{x}) = 0 \tag{2.24}$$

where $\boldsymbol{x} \in \mathbb{R}^n$ and $\boldsymbol{F} : \mathbb{R}^n \to \mathbb{R}^n$. The classic reference for this subject is [29]. Many of the methods for the solution of linear systems can be generalized or used to solve nonlinear systems.

2.3.1 Newton's Method

The most commonly used method for solving (2.24) is Newton's method, an iterative algorithm based upon linearization about successive solutions. Given any approximate solution $\boldsymbol{x}^k$, there exists some $\Delta\boldsymbol{x}^{k+1}$ that can be added to $\boldsymbol{x}^k$ such that

$$\boldsymbol{F}(\boldsymbol{x}^k + \Delta\boldsymbol{x}^{k+1}) = 0$$

Newton's method is based on setting the linear first-order Taylor expansion equal to zero and solving for $\Delta\boldsymbol{x}^{k+1}$, i.e.

$$\boldsymbol{F}(\boldsymbol{x}^k + \Delta\boldsymbol{x}^{k+1}) \approx \boldsymbol{F}(\boldsymbol{x}^k) + \boldsymbol{J}_F(\boldsymbol{x}^k)\,\Delta\boldsymbol{x}^{k+1} = 0,$$

where $\boldsymbol{J}_F(\boldsymbol{x}) \in \mathbb{R}^{n\times n}$ is the Jacobian derivative of $\boldsymbol{F}$. Because the function $\boldsymbol{F}$ is nonlinear, the first-order Taylor expansion is only an approximation, and the method must continue after setting $\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \Delta\boldsymbol{x}^{k+1}$. Thus the steps of a Newton iteration are:

1. solve for delta vector $\Delta\boldsymbol{x}^{k+1}$

$$\boldsymbol{J}_F(\boldsymbol{x}^k)\,\Delta\boldsymbol{x}^{k+1} = -\boldsymbol{F}(\boldsymbol{x}^k), \tag{2.25}$$

2. and update solution vector

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \Delta\boldsymbol{x}^{k+1}. \tag{2.26}$$

The advantage of Newton's method is that if the initial guess, $\boldsymbol{x}^0$, is sufficiently close to the exact solution, then Newton's method converges quadratically [29], i.e.

$$\left\|\boldsymbol{x}^{k+1} - \boldsymbol{x}^*\right\| \le c\left\|\boldsymbol{x}^k - \boldsymbol{x}^*\right\|^2,$$

where $\boldsymbol{x}^*$ denotes the exact solution. However, Newton's method requires a linear system solution step at each iteration. This can be expensive in terms of computation and in terms of storage, especially if a direct factorization method is used. Furthermore, merely generating the Jacobian matrix $\boldsymbol{J}_F(\boldsymbol{x}^k)$ and evaluating the function for the right hand side vector can be costly for complicated nonlinear functions. Finally, if the initial guess is not close enough to the exact solution then Newton's method can diverge or get caught in a limit cycle. Thus there exist many approximate Newton methods and damped Newton schemes [29, 3].

2.3.2 Nonlinear Relaxation

The relaxation-Newton iterative algorithms are a generalization of the idea of linear relaxation [29]. In a relaxation-Newton iteration, the n-dimensional equation system (2.24) is decomposed into as many as n smaller nonlinear sub-problems, each of which is solved independently with Newton's method. For example, in each iteration of the Gauss-Jacobi-Newton method, the i^{th} component of equation (2.24),

$$F_i(x_1^k, \ldots, x_{i-1}^k, x_i^{k+1}, x_{i+1}^k, \ldots, x_n^k) = 0$$

is solved for x_i^{k+1}, the i^{th} component of the unknown vector. Similarly, within each iteration of the Gauss-Seidel-Newton method, the equation

$$F_i(x_1^{k+1}, \ldots, x_{i-1}^{k+1}, x_i^{k+1}, x_{i+1}^k, \ldots, x_n^k) = 0$$

is solved for x_i^{k+1}. Just as in the case of linear relaxation, the essential difference between the Gauss-Jacobi and Gauss-Seidel algorithms is that, in the $k+1^{\text{st}}$ iteration, when computing x_i^{k+1}, the Gauss-Seidel algorithm uses the values of x_j^{k+1} for the solution of j^{th} subsystem if $j < i$. In other words, the Gauss-Seidel algorithm uses the most recently computed information that is available.

One simple way to reduce the computation cost of relaxation-Newton algorithms is to limit the number of Newton iterations when solving each sub-problem to a small fixed number m, such as $m = 1$ [29]. The justification for solving the sub-problems only approximately is that each nonlinear relaxation iteration is only approximately converged. It can be shown that this does not affect the convergence rate of a relaxation-Newton method. In addition, the block relaxation technique and successive overrelaxation may be used to reduce the number of nonlinear relaxation iterations required for convergence [29]. Provided that the relaxation converges quickly enough, an accelerated one-step relaxation-Newton method can be a highly efficient method for the solution of nonlinear problems.

2.4 Solution of Initial Value Problems

This section contains a brief review of two different approaches for solving the time-dependent nonlinear initial-value problem

$$\tfrac{d}{dt}\boldsymbol{x} + \boldsymbol{F}(\boldsymbol{x}(t), t) = 0 \quad \text{with} \quad \boldsymbol{x}(0) = \boldsymbol{x}_0, \tag{2.27}$$

where $\boldsymbol{x}(t) \in \mathbb{R}^n$ is the unknown, $\boldsymbol{x}_0$ is an initial condition, and $\boldsymbol{F} : \mathbb{R}^{n+1} \to \mathbb{R}^n$. The operator formulation of this IVP can be written as

$$\mathcal{F}\,\boldsymbol{x} = 0 \quad \text{with} \quad \boldsymbol{x}(0) = \boldsymbol{x}_0, \tag{2.28}$$

where $\boldsymbol{x}$ is a member of a function space and $\mathcal{F}$ is a nonlinear differential operator. It is assumed that the equation is to be solved numerically on a finite time interval $[0, T]$.

2.4.1 Pointwise Methods

The standard approach to numerically solving (2.27) is to begin with the known solution at time $t_0 = 0$, and then march through the interval $[0, T]$, timepoint by timepoint, generating the solution at successive discrete timepoints, with

$$t_{m+1} = t_m + h_{m+1}. \tag{2.29}$$

In the m^{th} step of this pointwise approach, a timestep h_m is selected, and the $\frac{d}{dt}$ term of the differential equation system (2.27) is discretized with a *multistep integration method.* This yields either an implicit or explicit system of nonlinear equations that is solved for $\boldsymbol{x}(t_{m+1})$.

Multistep Integration Methods

Suppose that a sequence of discretization timesteps $\{h_1, \ldots, h_m, \ldots\}$ and the corresponding sequence of timepoints $\{t_1, \ldots, t_{m+1}, \ldots, T\}$ have been selected. To solve (2.27), an s-step multistep integration method, and the values of $\boldsymbol{x}$ and $\boldsymbol{F}$ at the previous s timepoints are used to compute an approximation $\boldsymbol{x}_m$ to the value of $\boldsymbol{x}(t)$ at time $t = t_m$. The multistep equation generated at timepoint t_m is

$$\sum_{j=0}^{s} \alpha_j\, \boldsymbol{x}_{m-j} = \sum_{j=0}^{s} h_{m-j}\, \beta_j\, \boldsymbol{F}(\boldsymbol{x}_{m-j}, t_{m-j}), \tag{2.30}$$

where $\alpha_0 = 1$ and either $\alpha_s \neq 0$ or $\beta_s \neq 0$ [11, 9]. The other coefficients α_j and β_j are chosen to maintain accuracy and consistency within stability limits. Note that the coefficients α_j and β_j depend on m if the timesteps h_m are not constant.

There are many standard terms associated with multistep methods. A multistep method has *accuracy* of order p if it computes the solution of the initial value problem (2.27) exactly when the solution is a polynomial in time of degree less than or equal to p. A method is said to be *consistent* if it has order of accuracy $p \geq 1$. A method is *convergent* if it is possible to arbitrarily accurately approximate the exact solution to the

time-continuous differential equation system (2.27) uniformly over the simulation interval $[0, T]$ by reducing the discretization timestep.

There are several different types of stability. A multistep method with uniform timestep h is said to be *stable* if the computed values $\boldsymbol{x}(t)$ remain bounded as $h \to 0$. This definition of stability is leads to a well-known theorem that a multistep method is convergent if and only if it is consistent and stable. A multistep method is said to be *A-stable* if, when applied to the test differential equation

$$\frac{d}{dt}\boldsymbol{x} = \boldsymbol{A}\,\boldsymbol{x}(t) \quad \text{with} \quad \boldsymbol{x}(0) = \boldsymbol{x}_0, \tag{2.31}$$

where $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ and the eigenvalues of $\boldsymbol{A}$ all have negative real parts, the method leads to a bounded solution for any timestep size as the number of timesteps approaches infinity.

There are two classes of multistep methods. A multistep method is said to be *explicit* if in (2.30) it uses only the values $\boldsymbol{x}$ and $\boldsymbol{F}$ evaluated at previous timepoints when generating the solution at a timepoint, i.e. $\beta_0 = 0$. When $\beta_0 \neq 0$, the multistep method is said to be *implicit*, since in this case, an implicit equation in terms of $\boldsymbol{x}_m$ and $\boldsymbol{F}(\boldsymbol{x}_m, t_m)$ must be solved for $\boldsymbol{x}_m$. For nonlinear problems, the implicit equation is usually solved with Newton's method, requiring at least one matrix solve.

Explicit methods may seem to have a significant computational advantage over implicit methods because there is no need to perform a nonlinear system solution. However, explicit methods are far less stable. It is easily shown that implicit methods can be computationally superior to explicit methods in practice, especially for *stiff problems*, with eigenvalues that differ by several orders of magnitude. Such problems exhibit behavior on vastly different time scales. For a stiff problem, the stability of the implicit methods allows the use of substantially larger timesteps, resulting in lower overall computational work for the simulation.

A particularly useful class of implicit multistep methods are the backward difference formulas (BDF), characterized by $\beta_0 \neq 0$ but $\beta_1 = \cdots = \beta_s = 0$. The 1^{st}-order BDF (also known as the Backward Euler method) and the 2^{nd}-order BDF (also known as the 2^{nd}-order Gear method) are, respectively,

$$\boldsymbol{x}_m = \boldsymbol{x}_{m-1} + h_m\,\boldsymbol{F}(\boldsymbol{x}_m, t_m) \tag{2.32}$$

$$\boldsymbol{x}_m = \left(\frac{(h_m + h_{m-1})^2}{h_{m-1}(2h_m + h_{m-1})}\right)\boldsymbol{x}_{m-1} - \left(\frac{h_m^2}{h_{m-1}(2h_m + h_{m-1})}\right)\boldsymbol{x}_{m-2} + \left(\frac{h_m(h_m + h_{m-1})}{2h_m + h_{m-1}}\right)\boldsymbol{F}(\boldsymbol{x}_m, t_m). \tag{2.33}$$

In the uniform timestep case, these two BDF methods are known to be A-stable [11]. In the general nonuniform timestep case, it is possible to make the 2^{nd}-order BDF unstable by choosing rapidly varying successive timesteps. Nevertheless, it has been proven that the variable timestep 2^{nd}-order BDF is always stable in some sense as long as the ratio of successive timesteps h_m/h_{m-1} is kept below 1.2 [6]. In practice, a ratio limit of 2.0 is sufficient.

LTE and Timestep Control

Of course, multistep integration methods can provide only an approximate solution to the continuous problem (2.27). In practice, this unavoidable error in the numerical solution will be due to the machine error caused by finite precision floating-point arithmetic – but with a double-precision implementation this will typically make a small contribution. Primarily, the error in the numerical solution will be caused by the discrete-time approximation of the multistep formula itself. Not surprisingly, this truncation error of the multistep method is closely related to the order of accuracy of the method.

The *local truncation error* (LTE) of a multistep method is defined to be the result of plugging the exact solution $\boldsymbol{x}^*(t)$ of (2.27) into the multistep equation (2.30) [11, 16]

$$\text{LTE} \equiv \sum_{j=0}^{s} \alpha_j \, \boldsymbol{x}^*(t_{m-j}) - \sum_{j=0}^{s} h_{m-j} \, \beta_j \, \boldsymbol{F}(\boldsymbol{x}^*(t_{m-j}), t_{m-j}). \tag{2.34}$$

In other words, the LTE is the amount by which the solution of the differential equation fails to satisfy the equation used in the numerical method. If the localization assumption is made that the computed solution $\boldsymbol{x}(t)$ is precisely equal to the exact solution $\boldsymbol{x}^*(t)$ at all timepoints previous to t_m, then the LTE expression (2.34) may be rewritten as

$$\text{LTE} \equiv \boldsymbol{x}^*(t_m) - \boldsymbol{x}_m. \tag{2.35}$$

Note that neither of the LTE expressions (2.34) nor (2.35) may be computed, since the exact solution $\boldsymbol{x}^*$ is not available. Nevertheless, it is possible to closely approximate the LTE of a p^{th}-order multistep method at a timepoint by comparing the solution computed at that timepoint to a predicted value $\boldsymbol{x}_m^p$ computed by fitting a p^{th}-order polynomial to the values at the $p+1$ previous timepoints [25, 9, 39, 23, 38]. It has been shown in general that for a p^{th}-order BDF method, the LTE at time t_m is approximately given by

$$\text{LTE} \approx \frac{\beta_0}{t_m - t_{m-p-1}} \left(\boldsymbol{x}_m - \boldsymbol{x}_m^p \right) \tag{2.36}$$

where β_0 is the coefficient of the BDF multistep formula (2.30) [4, 5].

For a multistep method with order of accuracy p and uniform timestep h, the LTE is a term of $\mathcal{O}(h^{p+1})$. Perhaps more importantly, there are no A-stable multistep integration methods whose local truncation error is of an order higher than h^3 [9].

In practice, the timestep size for a variable-timestep multistep integration formula is usually determined with LTE timestep control [39, 2]. First a timestep is chosen, the value $\boldsymbol{x}_m$ is computed, and then $\boldsymbol{x}_m$ is compared to the value $\boldsymbol{x}_m^p$ predicted by fitting a polynomial to previously computed values. The difference between the computed value and this predicted value is used to estimate the LTE. If the LTE is too large, the timestep is rejected and the program tries again with a smaller timestep. Otherwise, the timestep and the computed value are accepted, and the program moves on to the next timestep.

2.4.2 Waveform Methods

The waveform methods for solving the nonlinear IVP (2.27) are extensions to function space of various numerical algorithms for solving nonlinear algebraic problems, such as those discussed in Section 2.3. Whereas Newton's method and the nonlinear relaxation algorithms operate on variables that are vectors in $\boldsymbol{x}^k \in \mathbb{R}^n$, waveform methods operate on vectors of waveforms on a closed time interval, $\boldsymbol{x}^k \in (\mathbb{R}^n, [0,T])$. As will be shown, waveform methods are easily parallelized.

Waveform Newton Method

The waveform Newton method (WN) is obtained by applying an abstracted form of Newton's method to (2.27) [15, 37]. To apply WN to equation (2.27), the differential equation itself is linearized with the Frechet derivative $\mathcal{J}_{\mathcal{F}}$ of $\mathcal{F}$ given by

$$\mathcal{J}_{\mathcal{F}}\, \boldsymbol{x}(t) = \tfrac{d}{dt}\, \boldsymbol{x} + \boldsymbol{J}_F(\boldsymbol{x}(t), t), \tag{2.37}$$

before time discretization with a multistep algorithm. First, an initial guess waveform $x_i^0 \in (\mathbb{R}, [0,T])$ satisfying initial conditions is chosen for each element of the vector of waveforms $\boldsymbol{x}^0 \in (\mathbb{R}^n, [0,T])$. Then the steps of a WN iteration are:

1. solve for the vector of waveforms $\Delta\boldsymbol{x}^{k+1}$

$$\mathcal{J}_{\mathcal{F}}(\boldsymbol{x}^k)\, \Delta\boldsymbol{x}^{k+1} = -\mathcal{F}(\boldsymbol{x}^k), \tag{2.38}$$

2. update the solution vector of waveforms at all $t \in [0,T]$

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \Delta\boldsymbol{x}^{k+1}. \tag{2.39}$$

After time discretization with a multistep method, the WN method reduces to the algebraic Newton method operating on a "large" vector consisting of the concatenation of vectors $\Delta \boldsymbol{x}^k[m]$ at all discrete timepoints, with the Frechet derivative corresponding to a "large" block lower-triangular matrix. For example, in the case of backward Euler, the first step of the WN iteration becomes the algebraic problem

$$\begin{bmatrix} \frac{1}{h}\boldsymbol{I}+\boldsymbol{J}_F(\boldsymbol{x}^k[1]) & & & & \\ -\frac{1}{h}\boldsymbol{I} & \frac{1}{h}\boldsymbol{I}+\boldsymbol{J}_F(\boldsymbol{x}^k[2]) & & & \\ & -\frac{1}{h}\boldsymbol{I} & \frac{1}{h}\boldsymbol{I}+\boldsymbol{J}_F(\boldsymbol{x}^k[3]) & & \\ & & \ddots & \ddots & \end{bmatrix} \begin{bmatrix} \Delta \boldsymbol{x}^{k+1}[1] \\ \Delta \boldsymbol{x}^{k+1}[2] \\ \Delta \boldsymbol{x}^{k+1}[3] \\ \vdots \end{bmatrix} = \begin{bmatrix} -\boldsymbol{F}(\boldsymbol{x}^k[1]) \\ -\boldsymbol{F}(\boldsymbol{x}^k[2]) \\ -\boldsymbol{F}(\boldsymbol{x}^k[3]) \\ \vdots \end{bmatrix}$$

where $\boldsymbol{J}_F(\boldsymbol{x}^k[m])$ denotes the Jacobian derivative of $\boldsymbol{F}$ evaluated at time point m. Note that each diagonal block $\frac{1}{h}\boldsymbol{I}+\boldsymbol{J}_F(\boldsymbol{x}^k[m])$ is the same as the matrix that is generated by using the algebraic Newton method to solve at each timestep with the pointwise direct method. Just like the algebraic Newton method, WN offers quadratic convergence provided that the waveform iterate is close enough to the exact solution. A more detailed discussion of the convergence properties of the waveform Newton method for circuit simulation problems can be found in [37].

Waveform Relaxation

The waveform relaxation (WR) method is a dynamic iteration process obtained by applying relaxation directly to the system of differential equations comprising the initial value problem (2.27) [18]. Also known as Picard-Lindelöf iteration [24], the WR method decomposes the IVP equation system into subsystems *before* time discretization with a multistep method. The solution of each differential subsystem is a *waveform* in time, and the original system is solved iteratively by solving the subsystems independently, using the waveform solutions from previous iterations for the variables from other subsystems. The WR algorithm is shown pictorially in Figure 2-4

More precisely, the Gauss-Jacobi WR and Gauss-Seidel WR algorithms for solving (2.27) are given in Algorithms 2.4.1 and 2.4.2. In iteration k of Gauss-Jacobi WR, the i^{th} equation of the n-dimensional differential equation system (2.27) is solved for waveform x_i^{k+1}, and the waveforms x_j^k of the previous iteration are used as solutions for the other components $j \neq i$. The algorithm for Gauss-Seidel WR is similar, with the exception that when computing x_i^{k+1}, the waveform x_j^{k+1} for the j^{th} subsystem is used if it has already been computed, otherwise the previous iteration waveform x_j^k is used.

The WR method has several computational advantages. Since it is an iterative method, WR avoids factoring and storing large matrices. Since the subsystems are

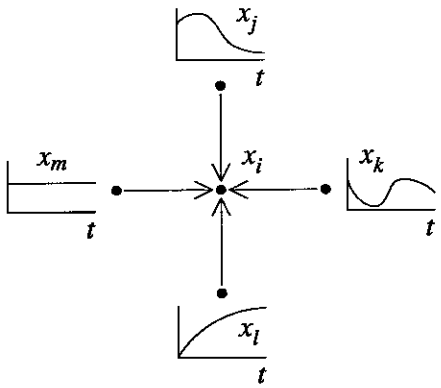

FIGURE 2-4: In each iteration of a WR algorithm, waveform x_i is computed for $t \in [0, T]$, while waveforms x_j, x_k, x_l, x_m are held fixed. Then x_j is computed, etc...

ALGORITHM 2.4.1 (GAUSS-JACOBI WR FOR SOLVING (2.27)).

1. *Initialize:* Pick vector of waveforms $\boldsymbol{x}^0$.
2. For $k = 0, 1, \ldots$ until converged

   For $i = 1$ to n, solve for x_i^{k+1} waveform:

$$\frac{d}{dt} x_i^{k+1}(t) + F_i(x_1^k(t), \ldots, x_{i-1}^k(t), x_i^{k+1}(t), x_{i+1}^k(t), \ldots, x_n^k(t), t) = 0$$

$$\text{with} \quad x_i^{k+1}(0) = x_{0_i}$$

solved independently, different sets of timepoints may be used to resolve the waveforms of different subsystems. This can be a significant computational advantage in a system exhibiting *multirate behavior*, in which the solutions of different subsystems change at different rates and/or at different times in the simulation interval, as shown in Figure 2-5. Finally, on a parallel machine, WR can be highly efficient since it leads to large, decoupled chunks of computation that can be easily parallelized.

Of course, WR is efficient relative to the standard pointwise direct method of solving (2.27) only if it converges in few enough iterations. For many problems, the WR algorithm 2.4.1 can be shown to have *guaranteed convergence* on a finite time interval $[0, T]$, starting from *any* initial guess that matches initial conditions [50]. Nevertheless, when applied to solving the device simulation equation system (1.9)–(1.11), the WR algorithm converges slowly, unless acceleration techniques such as block relaxation, conjugate directions, or successive overrelaxation are applied [21, 33].

ALGORITHM 2.4.2 (GAUSS-SEIDEL WR FOR SOLVING (2.27)).

1. *Initialize:* Pick vector of waveforms $\boldsymbol{x}^0$.

2. For $k = 0, 1, \ldots$ until converged

   For $i = 1$ to n, solve for x_i^{k+1} waveform:

$$\frac{d}{dt} x_i^{k+1}(t) + F_i(x_1^{k+1}(t), \ldots, x_{i-1}^{k+1}(t), x_i^{k+1}(t), x_{i+1}^{k}(t), \ldots, x_n^{k}(t), t) = 0$$

$$\text{with} \quad x_i^{k+1}(0) = x_{0_i}$$

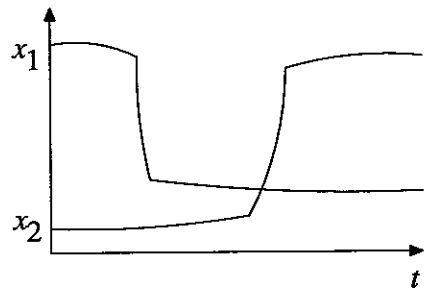

FIGURE 2-5: A system is said to exhibit *multirate* behavior if different components of the system change at different times and/or at different rates.

Waveform Relaxation Newton

The waveform relaxation Newton (WRN) method is one way to reduce the computation per WR iteration and improve the WR efficiency [50, 37]. Analogous to the one-step relaxation-Newton acceleration of nonlinear relaxation, in a WRN iteration, each subsystem is solved only approximately with one step of the waveform Newton method. The WRN method is implemented by computing only one algebraic Newton iteration at each timepoint, using a value from the previous waveform iterate as an initial guess. Like the one-step relaxation-Newton acceleration, the WRN acceleration does not affect the asymptotic convergence rate of WR. A more detailed discussion of the convergence properties of the WRN method can be found in [37, 50].

Another way to reduce the computational cost of each iteration is to take advantage of the waveform iterative nature of the WR algorithm and minimize the number of timepoints used to compute the waveform of each subsystem. With a *timestep refinement* strategy [37], relatively few timesteps are used to compute the waveforms in the early WR iterations. In subsequent WR iterations, additional timepoints are introduced only when they are required to keep the local truncation error in the waveform less than the

relaxation error.

3

WR Applied to Device Simulation

3.1 Introduction

An effective approach to solving the device equation system (1.9)–(1.11) with a parallel computer is to decompose the system into subsystems before time discretization with a waveform relaxation (WR) algorithm. The system is then solved iteratively by solving the subsystems independently, using fixed waveforms from previous iterations for the variables from other subsystems. The device WR algorithm is shown pictorially in Figure 3-1, and is given in Algorithm 3.1.1, where F_{1_i}–F_{3_i} are specified by (1.12)–(1.14).

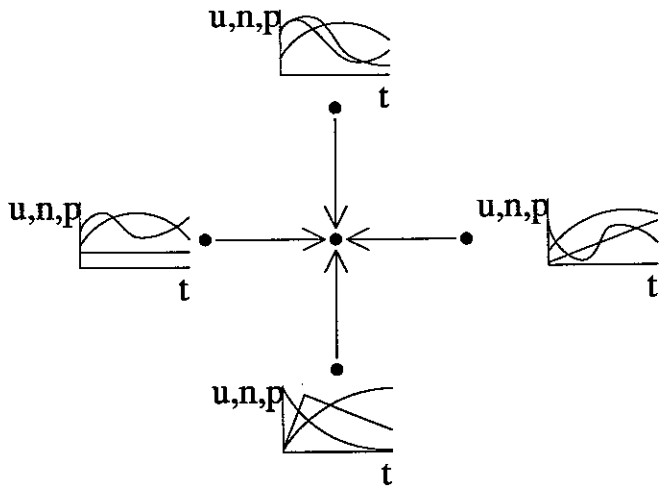

FIGURE 3-1: In node-by-node WR, solutions at a node are computed by holding the waveforms of neighboring mesh nodes fixed and solving a small 3×3 system of equations.

In each WR iteration, the $u(t)$, $n(t)$, and $p(t)$ waveforms of each mesh node i are computed by solving the equation system of the node, while holding the waveforms of the neighboring mesh nodes fixed. This reduces the problem of simultaneously solving

ALGORITHM 3.1.1 (GAUSS-JACOBI WR FOR DEVICE SIMULATION).

1. *Initialize:* Pick u^0, n^0, p^0 waveforms at all nodes.

2. For $k = 0, 1, \ldots$ until converged

   For each node i, solve for $u_i^{k+1}, n_i^{k+1}, p_i^{k+1}$ waveforms:

$$\begin{array}{rcl} & F_{1_i}(u_i^{k+1}(t), n_i^{k+1}(t), p_i^{k+1}(t), u_j^k(t), t) & = 0 \\ \frac{d}{dt} n_i^{k+1} + & F_{2_i}(u_i^{k+1}(t), n_i^{k+1(t)}, u_j^k(t), n_j^k(t), t) & = 0 \\ \frac{d}{dt} p_i^{k+1} + & F_{3_i}(u_i^{k+1}(t), p_i^{k+1}(t), u_j^k(t), p_j^k(t), t) & = 0 \end{array} \qquad (3.1)$$

$$\text{with} \quad u_i^{k+1}(0) = u_{0_i}, \quad n_i^{k+1}(0) = n_{0_i}, \quad p_i^{k+1}(0) = p_{0_i}$$

the large nonlinear differential-algebraic system of $3N$ equations to one of independently solving the smaller differential-algebraic system of 3 equations at each silicon node. Each small nonlinear system of 3 equations can be solved with Newton's method and an implicit numerical integration method such as the 2^{nd}-order backward difference formula.

The following sections describe the convergence of ordinary WR for device simulation. Although a global convergence result will be shown, it should be kept in mind that when applied to solving the device simulation equation system (1.9)–(1.11), the device WR algorithm converges slowly, unless acceleration techniques such as block relaxation, conjugate directions, or successive overrelaxation are applied [21, 33, 31].

3.2 WR Convergence

The following device WR convergence theorem will show that, when used to solve the device equation system on a finite time interval $[0, T]$, the device WR Algorithm 3.1.1 has *guaranteed convergence* starting from *any* initial guess that matches initial conditions $\boldsymbol{u}_0, \boldsymbol{n}_0, \boldsymbol{p}_0$. The theorem differs from a similar theorem for circuit simulation WR [18, 50]. At iteration $k+1$ of device WR, the solutions at node i of the differential variables n_i^{k+1} and p_i^{k+1} depend upon the solution of the algebraic variable u_i^{k+1}. The coupled solution at each mesh node must be computed by solving the coupled 3×3 differential-algebraic system (3.1). For the circuit simulation proof in [18], it is assumed that the variables of the WR equation system can be indexed such that the solutions of the differential variables $\boldsymbol{x}^{k+1}$ do not depend on the solutions of the algebraic variables $\boldsymbol{z}^{k+1}$ at the same iteration.

Like the convergence theorem for circuit simulation, the device WR theorem requires the definition of a special norm on a waveform.

DEFINITION 3.2.1. *The β-norm $\|\boldsymbol{x}\|_B$ on a waveform $\boldsymbol{x} : [0,T] \rightarrow \mathbb{R}^n$ is defined as*

$$\|\boldsymbol{x}\|_B \equiv \max_{t\in[0,T]} e^{-Bt} \|\boldsymbol{x}(t)\| \quad \textit{where } B \in \mathbb{R} \textit{ and } B > 0$$

In the following, by first considering the Poisson equation (1.9) and then the electron and hole transport equations (1.10)–(1.11), it is shown that the effect of 3×3 block WR on the device simulation system may be characterized by a 3×3 system of inequalities, relating norms of the changes in the waveforms over successive WR iterations. The contraction mapping theorem is then invoked to prove convergence.

THEOREM 3.2.2 (CONVERGENCE OF NODE-BY-NODE WR).
For any finite time interval $t \in [0,T]$, and for any initial guess waveforms $\boldsymbol{u}^0(t)$, $\boldsymbol{n}^0(t)$ and $\boldsymbol{p}^0(t)$ that match the initial conditions, the sequence of waveforms produced by the node-by-node Gauss-Jacobi WR Algorithm 3.1.1 will converge to the exact solution of the device simulation DAE system (1.9)–(1.11).

Proof. Suppose that the node-by-node WR algorithm 3.1.1 is applied to a device spatially discretized on a mesh of N nodes. At timepoint $t \in [0,T]$ in the k^{th} iteration, the iteration equations of the node-by-node WR algorithm 3.1.1 may be written in terms of vectors $\boldsymbol{u}(t), \boldsymbol{n}(t), \boldsymbol{p}(t) \in \mathbb{R}^N$ as

$$\begin{aligned}
\boldsymbol{D}\,\boldsymbol{u}^k(t) + (\boldsymbol{L}+\boldsymbol{U})\,\boldsymbol{u}^{k-1}(t) + \boldsymbol{E}\,\boldsymbol{n}^k(t) - \boldsymbol{E}\,\boldsymbol{p}^k(t) - \boldsymbol{b} &= 0 && (3.2)\\
\tfrac{d}{dt}\boldsymbol{n}^k(t) + \boldsymbol{F}_2\left(\boldsymbol{u}^k(t), \boldsymbol{n}^k(t), \boldsymbol{u}^{k-1}(t), \boldsymbol{n}^{k-1}(t)\right) &= 0 && (3.3)\\
\tfrac{d}{dt}\boldsymbol{p}^k(t) + \boldsymbol{F}_3\left(\boldsymbol{u}^k(t), \boldsymbol{p}^k(t), \boldsymbol{u}^{k-1}(t), \boldsymbol{p}^{k-1}(t)\right) &= 0 && (3.4)
\end{aligned}$$

where $\boldsymbol{A} = \boldsymbol{D}-\boldsymbol{L}-\boldsymbol{U}$ is the standard Gauss-Jacobi relaxation splitting of the irreducibly diagonally dominant and symmetric matrix $\boldsymbol{A}$ given by the potential dependent portions of the Poisson equations, $\boldsymbol{E} \in \mathbb{R}^{N\times N}$ is a diagonal matrix given by the node areas, $\boldsymbol{b} \in \mathbb{R}^N$ is a vector given by the product of the node areas and the node dopings, and $\boldsymbol{F}_2$ and $\boldsymbol{F}_3$ are Lipschitz continuous nonlinear algebraic functions.

Subtracting successive iterations of the Poisson equation (3.2) and rearranging terms:

$$\delta\boldsymbol{u}^k(t) = \boldsymbol{C}\,\delta\boldsymbol{p}^k(t) - \boldsymbol{C}\,\delta\boldsymbol{n}^k(t) + \boldsymbol{M}\,\delta\boldsymbol{u}^{k-1}(t) \qquad (3.5)$$

where $\boldsymbol{M} = \boldsymbol{D}^{-1}(\boldsymbol{L}+\boldsymbol{U})$ is the Gauss-Jacobi iteration matrix of the Poisson equation matrix $\boldsymbol{A}$ and $\boldsymbol{C} = \boldsymbol{D}^{-1}\boldsymbol{E}$. Since $\boldsymbol{A}$ is irreducibly diagonally dominant, the spectral

radius of $\boldsymbol{M}$ is strictly less than unity [44]. Therefore, there exists a norm $\|\cdot\|$ for which the induced norm $\|\boldsymbol{M}\| < 1$ [48, 29]. Since $\boldsymbol{A}$ is symmetric, the matrix $\boldsymbol{M}$ is diagonalizable, i.e. $\boldsymbol{M} = \boldsymbol{S}^{-1}\Lambda\boldsymbol{S}$ where $\boldsymbol{S}$ is a matrix of the eigenvectors of $\boldsymbol{M}$ and Λ is a diagonal matrix of the corresponding eigenvalues [28]. Therefore, the S-norm defined as $\|\boldsymbol{x}\|_S = \|\boldsymbol{S}\boldsymbol{x}\|_1$ for $\boldsymbol{x} \in \mathbb{R}^N$ is an example of a norm for which the associated induced norm $\|\boldsymbol{M}\| < 1$, since

$$\|\boldsymbol{M}\|_S = \left\|\boldsymbol{S}\boldsymbol{M}\boldsymbol{S}^{-1}\right\|_1 = \left\|\boldsymbol{S}\cdot\boldsymbol{S}^{-1}\Lambda\boldsymbol{S}\cdot\boldsymbol{S}^{-1}\right\|_1 = \|\Lambda\|_1 = \rho(\boldsymbol{M}) < 1.$$

Taking the S-norm of both sides of equation (3.5) yields

$$\left\|\delta\boldsymbol{u}^k(t)\right\|_S \le c\left\|\delta\boldsymbol{p}^k(t)\right\|_S + c\left\|\delta\boldsymbol{n}^k(t)\right\|_S + m\left\|\delta\boldsymbol{u}^{k-1}(t)\right\|_S, \tag{3.6}$$

where $c = \|\boldsymbol{C}\|_S$ and $m = \|\boldsymbol{M}\|_S < 1$.

Integrating the electron transport equation (3.3), subtracting successive iterations, taking the S-norm of both sides, and invoking Lipschitz continuity yields

$$\begin{aligned}\left\|\delta\boldsymbol{n}^k(t)\right\|_S \;\le\; & \\ & \int_0^t \left\|\boldsymbol{F}_2\left(\boldsymbol{u}^k(\tau),\boldsymbol{n}^k(\tau),\boldsymbol{u}^{k-1}(\tau),\boldsymbol{n}^{k-1}(\tau)\right) - \boldsymbol{F}_2\left(\boldsymbol{u}^{k-1}(\tau),\boldsymbol{n}^{k-1}(\tau),\boldsymbol{u}^{k-2}(\tau),\boldsymbol{n}^{k-2}(\tau)\right)\right\|_S d\tau \\ & \le\; \ell_2\int_0^t\left\{\left\|\delta\boldsymbol{u}^k(\tau)\right\|_S + \left\|\delta\boldsymbol{n}^k(\tau)\right\|_S + \left\|\delta\boldsymbol{u}^{k-1}(\tau)\right\|_S + \left\|\delta\boldsymbol{n}^{k-1}(\tau)\right\|_S\right\}d\tau,\end{aligned}$$

where $\ell_2 > 0$ is a number related to the Lipschitz constant of $\boldsymbol{F}_2$ [29].

Applying Definition 3.2.1 of the β-norm to the inequality, multiplying and dividing by $e^{-B\tau}$ inside the integral yields:

$$\left\|\delta\boldsymbol{n}^k(t)\right\|_S \le \left\{\left\|\delta\boldsymbol{u}^k\right\|_{B_S} + \left\|\delta\boldsymbol{n}^k\right\|_{B_S} + \left\|\delta\boldsymbol{u}^{k-1}\right\|_{B_S} + \left\|\delta\boldsymbol{n}^{k-1}\right\|_{B_S}\right\}\ell_2\int_0^t e^{B\tau}d\tau,$$

where subscript B_S is used to denote the β-norm generated from the S-norm. Taking the β-norm of both sides and noting that

$$e^{-Bt}\int_0^t e^{B\tau}d\tau = e^{-Bt}\left[B^{-1}e^{B\tau}\Big|_0^t\right] = B^{-1}\left(1-e^{-Bt}\right) < B^{-1},$$

results in the expression

$$\left\|\delta\boldsymbol{n}^k\right\|_{B_S} \le \ell_2B^{-1}\left\|\delta\boldsymbol{u}^k\right\|_{B_S} + \ell_2B^{-1}\left\|\delta\boldsymbol{n}^k\right\|_{B_S} + \ell_2B^{-1}\left\|\delta\boldsymbol{u}^{k-1}\right\|_{B_S} + \ell_2B^{-1}\left\|\delta\boldsymbol{n}^{k-1}\right\|_{B_S}.$$

For $B > \ell_2$, this may be rewritten as

$$\left\|\delta\boldsymbol{n}^k\right\|_{B_S} \le \alpha_2\left\|\delta\boldsymbol{u}^k\right\|_{B_S} + \alpha_2\left\|\delta\boldsymbol{u}^{k-1}\right\|_{B_S} + \alpha_2\left\|\delta\boldsymbol{n}^{k-1}\right\|_{B_S} \quad \text{with} \quad \alpha_2 = \frac{\ell_2B^{-1}}{1-\ell_2B^{-1}}. \tag{3.7}$$

Note that no matter how large the number ℓ_2, there exists a finite B such that if B is chosen to be sufficiently large ($B > 2\ell_2$) then $\alpha_2 < 1$, and α_2 can be made as small as desired by further increasing B. In other words, as B is increased, the β-norm focuses on shorter and shorter intervals in $[0, T]$.

The hole transport equation (3.4) may be transformed into a similar inequality,

$$\left\|\delta \boldsymbol{p}^k\right\|_{B_S} \leq \alpha_3 \left\|\delta \boldsymbol{u}^k\right\|_{B_S} + \alpha_3 \left\|\delta \boldsymbol{u}^{k-1}\right\|_{B_S} + \alpha_3 \left\|\delta \boldsymbol{p}^{k-1}\right\|_{B_S} \quad \text{with} \quad \alpha_3 = \frac{\ell_3 B^{-1}}{1 - \ell_3 B^{-1}}, \tag{3.8}$$

where $\ell_3 > 0$ is a number related to the Lipschitz constant of f_3, and $B > \ell_3$.

Transforming the Poisson inequality (3.6) into an inequality using β-norms, and grouping it with the inequalities (3.7)-(3.8), where the β-norm is chosen so that

$$B > \max(\ell_2, \ell_3) \quad \text{and} \quad \alpha = \max(\alpha_2, \alpha_3) = \frac{\max(\ell_2, \ell_3) B^{-1}}{1 - \max(\ell_2, \ell_3) B^{-1}}$$

leads to the inequality system:

$$\begin{aligned}
\left\|\delta \boldsymbol{u}^k\right\|_{B_S} &\leq c \left\|\delta \boldsymbol{p}^k\right\|_{B_S} + c \left\|\delta \boldsymbol{n}^k\right\|_{B_S} + m \left\|\delta \boldsymbol{u}^{k-1}\right\|_{B_S} \\
\left\|\delta \boldsymbol{n}^k\right\|_{B_S} &\leq \alpha \left\|\delta \boldsymbol{u}^k\right\|_{B_S} + \alpha \left\|\delta \boldsymbol{u}^{k-1}\right\|_{B_S} + \alpha \left\|\delta \boldsymbol{n}^{k-1}\right\|_{B_S} \\
\left\|\delta \boldsymbol{p}^k\right\|_{B_S} &\leq \alpha \left\|\delta \boldsymbol{u}^k\right\|_{B_S} + \alpha \left\|\delta \boldsymbol{u}^{k-1}\right\|_{B_S} + \alpha \left\|\delta \boldsymbol{p}^{k-1}\right\|_{B_S},
\end{aligned}$$

which may be written in matrix form as

$$\begin{bmatrix} 1 & -c & -c \\ -\alpha & 1 & 0 \\ -\alpha & 0 & 1 \end{bmatrix} \begin{bmatrix} \|\delta \boldsymbol{u}^k\|_{B_S} \\ \|\delta \boldsymbol{n}^k\|_{B_S} \\ \|\delta \boldsymbol{p}^k\|_{B_S} \end{bmatrix} \leq \begin{bmatrix} m & 0 & 0 \\ \alpha & \alpha & 0 \\ \alpha & 0 & \alpha \end{bmatrix} \begin{bmatrix} \|\delta \boldsymbol{u}^{k-1}\|_{B_S} \\ \|\delta \boldsymbol{n}^{k-1}\|_{B_S} \\ \|\delta \boldsymbol{p}^{k-1}\|_{B_S} \end{bmatrix}.$$

The inverse of the left-hand side matrix is given by

$$\begin{bmatrix} 1 & -c & -c \\ -\alpha & 1 & 0 \\ -\alpha & 0 & 1 \end{bmatrix}^{-1} = \frac{1}{1 - 2\alpha c} \begin{bmatrix} 1 & c & c \\ \alpha & 1 - \alpha c & \alpha c \\ \alpha & \alpha c & 1 - \alpha c, \end{bmatrix},$$

so that if α is chosen to be sufficiently small ($\alpha < 1/(2c)$, which can be accomplished by using a β-norm with a large enough B), then the inverse is a positive matrix. Under this condition, both sides may be multiplied by the inverse without disturbing the inequality so that

$$\begin{bmatrix} \left\|\delta \boldsymbol{u}^k\right\|_{B_S} \\ \left\|\delta \boldsymbol{n}^k\right\|_{B_S} \\ \left\|\delta \boldsymbol{p}^k\right\|_{B_S} \end{bmatrix} \leq \frac{1}{1 - 2\alpha c} \begin{bmatrix} m + 2\alpha c & \alpha c & \alpha c \\ \alpha m + \alpha & \alpha - \alpha^2 c & \alpha^2 c \\ \alpha m + \alpha & \alpha^2 c & \alpha - \alpha^2 c \end{bmatrix} \begin{bmatrix} \left\|\delta \boldsymbol{u}^{k-1}\right\|_{B_S} \\ \left\|\delta \boldsymbol{n}^{k-1}\right\|_{B_S} \\ \left\|\delta \boldsymbol{p}^{k-1}\right\|_{B_S} \end{bmatrix}. \tag{3.9}$$

Defining $\boldsymbol{x} = [\boldsymbol{u}\ \boldsymbol{n}\ \boldsymbol{p}]^T$ and taking the infinity norm of both sides yields:

$$\left\|\delta\boldsymbol{x}^k\right\|_{B_S} \leq \gamma \left\|\delta\boldsymbol{x}^{k-1}\right\|_{B_S} \quad \text{where} \quad \gamma = \frac{1}{1-2\alpha c} \max\left(m + 4\alpha c\ ,\ \alpha m + 2\alpha\right).$$

If B is chosen large enough so that

$$\alpha = \frac{\max(\ell_2, \ell_3)B^{-1}}{1 - \max(\ell_2, \ell_3)B^{-1}} < \min\left(\frac{1-m}{6c}\ ,\ \frac{1}{m+2c+2}\right),$$

then the maximum row sums of T are less than one and

$$\left\|\delta\boldsymbol{x}^k\right\|_{B_S} \leq \gamma \left\|\delta\boldsymbol{x}^{k-1}\right\|_{B_S} \quad \text{with} \quad \gamma < 1. \tag{3.10}$$

Since there exists a norm for which inequality (3.10) holds with $\gamma < 1$, the node-by-node WR Algorithm 3.1.1 is a contraction mapping in an exponentially-weighted norm. Therefore, the contraction mapping theorem implies that the waveforms $\boldsymbol{x}^{k-1}$ must converge to the unique fixed-point of the WR device simulation system [29]. □

3.3 Sup-norm Convergence

Although Theorem 3.2.2 guarantees convergence on a finite time interval $[0, T]$, it does not imply fast convergence or computational efficiency. On the contrary, because of the use of the exponentially-weighted norm $\|\cdot\|_B$, the theorem shows only that as WR iterations proceed, convergence is guaranteed first on a small time interval $[0, \Delta t]$. Once this first interval is converged, WR will converge on the next interval $[\Delta t, 2\Delta t]$, and so on, until the entire interval $[0, T]$ is computed. In the early iterations, the latter parts of the waveforms may not move any closer to the exact solution.

Obviously, WR is most efficient if there is no computation wasted on timepoints whose values are not approaching the solution. And in practice, when WR is applied to the device simulation problem, the errors at all timepoints in the interval $[0, T]$ are reduced with each WR iteration. To demonstrate this, that the WR algorithm contracts in a sup-norm sense for typical MOSFET simulations, an example MOSFET simulation was performed over an interval of 30 nanoseconds with a short low-high-low voltage pulse applied to the drain with the gate held high. Figure 3-3 shows the electron concentration waveforms for several different WR iterations at a silicon-oxide interface node halfway between the source and drain. The initial guess was a flat waveform at the value satisfying the $t = 0$ conditions. The waveforms converge uniformly to the exact solution, i.e. values at all timepoints move towards the solution as iterations progress, so that no computation is wasted.

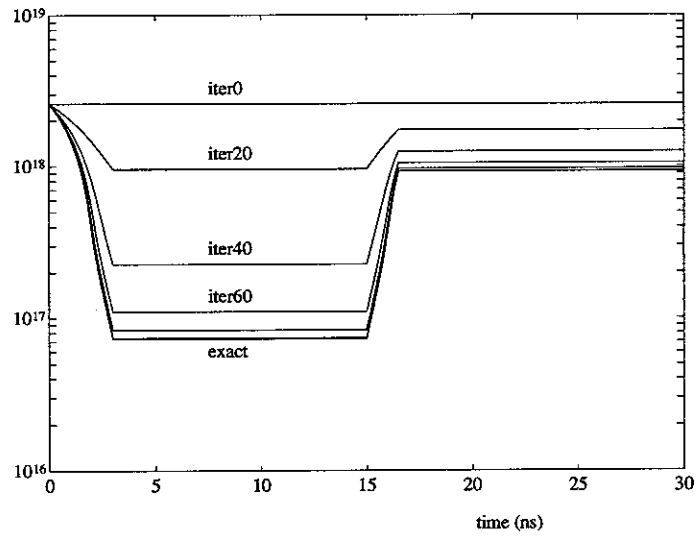

FIGURE 3-2: Sup-norm contractivity of electron concentration waveforms at a channel node.

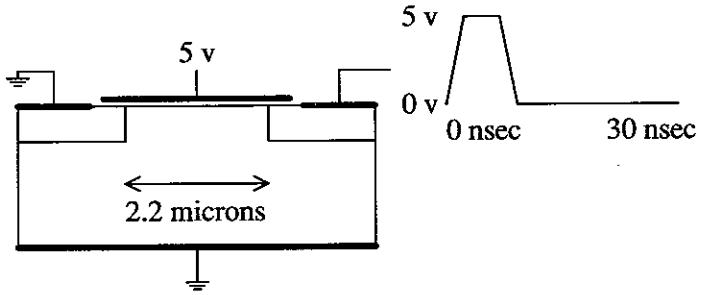

FIGURE 3-3: Experimental setup for demonstration of sup-norm contractivity.

The sup-norm contractivity of WR for a simplified 1D device simulation problem can be proven given strict conditions on the shape of the electric field [34, 32]. Consider a 1D evenly-spaced mesh of N nodes with the endpoint potentials $u_0(t)$ and $u_{N+1}(t)$ as inputs, as shown in Figure 3-4. If the potential $\boldsymbol{u}$ is known for all nodes, then the linear

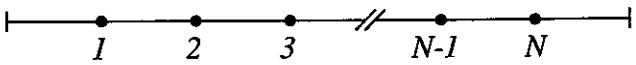

FIGURE 3-4: A one dimensional mesh consisting of N evenly-spaced nodes and boundary conditions at either end.

time-varying differential equation in $\boldsymbol{n}$ that arises at node i is

$$\frac{d}{dt} n_i^{k+1} + \frac{1}{l^2} \frac{kT\mu_n}{q} \sum_j \left[n_i^{k+1} B(u_i - u_j) - n_j^k B(u_j - u_i) \right] = 0 \qquad (3.11)$$

where l is the distance from node to node and the summation occurs over the two nodes to the left and right ($j = i \pm 1$). The following lemma and theorem hold.

LEMMA 3.3.1. *Given a sequence of N irreducible, positive matrices*

$$\{\boldsymbol{Q}(k), \boldsymbol{Q}(k+1), \ldots, \boldsymbol{Q}(k+N)\},$$

with each matrix $\boldsymbol{Q}(\cdot) \in \mathbb{R}^{N \times N}$ having an induced norm $\|\boldsymbol{Q}(\cdot)\|_\infty \leq 1$ and a row sum $\sum_j \boldsymbol{Q}(\cdot)_{ij} < 1$ for some row i, the product of the matrices satisfies

$$\|\boldsymbol{Q}(k+N)\, \boldsymbol{Q}(k+N-1) \cdots \boldsymbol{Q}(k)\|_\infty \leq \gamma < 1, \tag{3.12}$$

for some γ.

Proof. To show inequality (3.12), note that the product matrix

$$\boldsymbol{R} = \boldsymbol{Q}(k+N)\, \boldsymbol{Q}(k+N-1) \cdots \boldsymbol{Q}(k)$$

is a positive matrix since it is the product of positive matrices. Furthermore, for any positive matrix $\boldsymbol{T} \in \mathbb{R}^{N \times N}$, the vector norm $\|\boldsymbol{T}\boldsymbol{x}\|_\infty$ is maximized over all vectors $\|\boldsymbol{x}\|_\infty$ by a vector $\hat{\boldsymbol{x}}$ consisting of all ones. This implies that the induced norm is

$$\|\boldsymbol{R}\|_\infty = \|\boldsymbol{R}\hat{\boldsymbol{x}}\|_\infty . \tag{3.13}$$

Now suppose that the row i_k of $\boldsymbol{Q}(k)$ has a row sum less than one. By assumption, $\boldsymbol{Q}(k)$ is positive and $\|\boldsymbol{Q}(k)\|_\infty \leq 1$, so that row i_k of the matrix-vector product $\boldsymbol{Q}(k)\hat{\boldsymbol{x}}$ must be strictly less than one. All other rows of $\boldsymbol{Q}(k)\hat{\boldsymbol{x}}$ must be less than or equal to one.

The vector $\boldsymbol{Q}(k+1)\boldsymbol{Q}(k)\hat{\boldsymbol{x}}$ has at least two rows strictly less than one. To see this, suppose that row i_{k+1} of $\boldsymbol{Q}(k+1)$ has row sum less than one. If $i_{k+1} \neq i_k$ then both rows i_{k+1} and i_k of the matrix-vector product $\boldsymbol{Q}(k+1)\boldsymbol{Q}(k)\hat{\boldsymbol{x}}$ will be less than one. If $i_{k+1} = i_k$, then certainly row i_k of $\boldsymbol{Q}(k+1)\boldsymbol{Q}(k)\hat{\boldsymbol{x}}$ will be less than one. In addition, because $\boldsymbol{Q}(k+1)$ is irreducible, there must be at least one other row of $\boldsymbol{Q}(k+1)$ that has a non-zero element in *column* i_k. This row of $\boldsymbol{Q}(k+1)\boldsymbol{Q}(k)\hat{\boldsymbol{x}}$ will be less than one. In turn, each successive multiplication by one of the $\boldsymbol{Q}(\cdot)$ matrices reduces at least one more row. After N multiplications, the resulting matrix-vector product $\boldsymbol{R}\hat{\boldsymbol{x}}$ must have all rows less than one, proving the lemma. □

THEOREM 3.3.2 (SUP-NORM CONTRACTIVITY OF WR).
If at each time t, $\boldsymbol{u}(t)$ is such that the electric field along the line is either constant, or

monotonically decreasing, then WR applied to (3.11) is a contraction in a uniform norm on any finite interval $[0,T]$. *That is,*

$$\max_{t\in[0,T]} \left\| \Delta \boldsymbol{n}^{k+N}(t) \right\|_\infty \leq \gamma \max_{t\in[0,T]} \left\| \Delta \boldsymbol{n}^{k}(t) \right\|_\infty \tag{3.14}$$

where $\gamma < 1$, *superscript* k *denotes iteration,* N *is the number of grid points, and* $\Delta \boldsymbol{n}^k(t) = \boldsymbol{n}^k(t) - \boldsymbol{n}^{k-1}(t)$.

Proof. The WR iteration equations (3.11) applied to the model problem can be written in terms of unknown vector $\boldsymbol{n}(t)$ as

$$\tfrac{d}{dt}\, \boldsymbol{n}^{k+1}(t) = -\boldsymbol{D}(t)\, \boldsymbol{n}^{k+1}(t) + \boldsymbol{M}(t)\, \boldsymbol{n}^{k}(t) + \boldsymbol{b}(t) \tag{3.15}$$

where $\boldsymbol{D}(t) \in \mathbb{R}^{N\times N}$ is a diagonal matrix, $\boldsymbol{M}(t) \in \mathbb{R}^{N\times N}$ and $\boldsymbol{b}(t) \in \mathbb{R}^N$ accounts for the edge conditions u_0 and u_{N+1}. Because the Bernoulli function is always positive, both $\boldsymbol{D}(t)$ and $\boldsymbol{M}(t)$ are positive matrices.

The conditions on the electric field imply that at each node i,

$$u_i - u_{i-1} \leq u_{i+1} - u_i,$$

so that the monotonically decreasing Bernoulli function has values

$$B(u_i - u_{i-1}) \geq B(u_{i+1} - u_i) \quad \text{and} \quad B(u_{i-1} - u_i) \leq B(u_i - u_{i+1}).$$

Therefore, elements $d_{ii}(t)$ of $\boldsymbol{D}(t)$ and $m_{ij}(t)$ of $\boldsymbol{M}(t)$ will satisfy the relation

$$d_{ii}(t) \geq \epsilon_i + \sum_{j\neq i} m_{ij}(t) \tag{3.16}$$

where $\epsilon_i \geq 0$ and is strictly greater than zero for the mesh points $i = 1$ and $i = N$ next to the boundaries.

Given relationship (3.16) between $\boldsymbol{D}(t)$ and $\boldsymbol{M}(t)$, the WR algorithm applied to a system of the form of (3.15) will contract in a uniform norm. This has been shown for the case when $\boldsymbol{D}(t)$ and $\boldsymbol{M}(t)$ are independent of t, using Laplace transforms [24]. In the time dependent case, the result can be shown by examining the difference between iteration k and $k+1$ of (3.15), so that for each mesh point i,

$$\tfrac{d}{dt}\, \Delta n_i^{k+1}(t) = -d_{ii}(t)\, \Delta n_i^{k+1}(t) + \sum_{j\neq i} m_{ij}(t)\, \Delta n_j^{k}(t), \tag{3.17}$$

where $\Delta n_i^k(t) = n_i^k(t) - n_i^{k-1}(t)$. By assumption, $d_{ii}(t), m_{ij}(t) > 0$ and $\Delta n_i^k(0) = 0$ for all nodes i.

The solution $\Delta n_i^{k+1}(t)$ to equation (3.17) begins at value $\Delta n_i^{k+1}(0) = 0$ and throughout the entire interval $[0, T]$, stays within the bounds given by

$$\max_{t\in[0,T]} \left|\Delta n_i^{k+1}(t)\right| \leq \max_{t\in[0,T]} \left[\sum_{j\neq i} \frac{m_{ij}(t)}{d_{ii}(t)} \left|\Delta n_j^k(t)\right|\right] = L. \tag{3.18}$$

In other words, for any solution which violates this inequality, $\frac{d}{dt}\Delta n_i^{k+1}(t)$ points back into the region [49]. To see this, suppose that at some timepoint t_1, the solution $\Delta n_i^{k+1}(t)$ has moved from within the region to reach the upper limit L, then

$$\Delta n_i^{k+1}(t_1) = L \geq \sum_{j\neq i} \frac{m_{ij}(t_1)}{d_{ii}(t_1)} \left|\Delta n_j^k(t_1)\right| \geq \sum_{j\neq i} \frac{m_{ij}(t_1)}{d_{ii}(t_1)} \Delta n_j^k(t_1), \tag{3.19}$$

and therefore

$$d_{ii}(t_1)\, \Delta n_i^{k+1}(t_1) \geq \sum_{j\neq i} m_{ij}(t_1)\, \Delta n_j^k(t_1). \tag{3.20}$$

This implies that the right-hand side of equation (3.17) at time t_1 and the time derivative $\frac{d}{dt}\Delta n_i^{k+1}(t_1)$ are non-positive, prohibiting Δn_i^{k+1} from moving above the upper limit. Similarly, if at some timepoint t_2, the solution $\Delta n_i^{k+1}(t)$ has moved from within the region to reach the lower limit $-L$, then

$$\Delta n_i^{k+1}(t_2) = -L \leq -\sum_{j\neq i} \frac{m_{ij}(t_2)}{d_{ii}(t_2)} \left|\Delta n_j^k(t_2)\right| \leq 0, \tag{3.21}$$

and

$$-\, d_{ii}(t_2)\Delta n_i^{k+1}(t_2) \geq \sum_{j\neq i} m_{ij}(t_2) \left|\Delta n_j^k(t_2)\right| \geq \sum_{j\neq i} m_{ij}(t_2)\, \Delta n_j^k(t_2). \tag{3.22}$$

This implies that the right-hand side of equation (3.17) at time t_2 and the time derivative $\frac{d}{dt}\Delta n_i^{k+1}(t_2)$ are non-negative, prohibiting Δn_i^{k+1} from moving below the lower limit.

Written as a inner-product of vectors, inequality (3.18) implies that

$$\max_{t\in[0,T]} \left|\Delta n_i^{k+1}(t)\right| \leq \sum_{j\neq i} \frac{m_{ij}(\tau_{k,i})}{d_{ii}(\tau_{k,i})} \left|\Delta n_j^k(\tau_{k,i})\right| \tag{3.23}$$

$$\leq \left[\, \frac{m_{i1}(\tau_{k,i})}{d_{ii}(\tau_{k,i})} \;, \cdots, \; \frac{m_{iN}(\tau_{k,i})}{d_{ii}(\tau_{k,i})} \,\right] \begin{bmatrix} \max_{t\in[0,T]} \left|\Delta n_1^k(t)\right| \\ \vdots \\ \max_{t\in[0,T]} \left|\Delta n_N^k(t)\right| \end{bmatrix}, \tag{3.24}$$

where $\tau_{k,i}$ is the value of $t \in [0, T]$ which maximizes

$$\sum_{j\neq i} \frac{m_{ij}(t)}{d_{ii}(t)} \left|\Delta n_j^k(t)\right|. \tag{3.25}$$

Assembling the inequality system from (3.24) results in

$$\begin{bmatrix} \max\limits_{t\in[0,T]} \left|\Delta n_1^{k+1}(t)\right| \\ \vdots \\ \max\limits_{t\in[0,T]} \left|\Delta n_N^{k+1}(t)\right| \end{bmatrix} \leq \boldsymbol{Q}(k) \begin{bmatrix} \max\limits_{t\in[0,T]} \left|\Delta n_1^{k}(t)\right| \\ \vdots \\ \max\limits_{t\in[0,T]} \left|\Delta n_N^{k}(t)\right| \end{bmatrix} \tag{3.26}$$

where $\leq$ denotes a row-by-row inequality. The matrix $\boldsymbol{Q}(k) \in \mathbb{R}^{N\times N}$ is a positive matrix with elements given by

$$q_{ii}(k) = 0 \quad \text{and} \quad q_{ij}(k) = \frac{m_{ij}(\tau(k,i))}{d_{ii}(\tau(k,i))}.$$

Because of inequality (3.16), all rows of the matrices $\boldsymbol{Q}(k)$ have row sums less than or equal to 1, so that $\|\boldsymbol{Q}(k)\|_\infty \leq 1$. Because of the boundary conditions of the one dimensional mesh, the two rows $i = 1$ and $i = N$ of the matrix $\boldsymbol{Q}(k)$ have row sums strictly less than one. In addition, $\boldsymbol{Q}(k)$ is irreducible. Since these properties hold for all iterations k, Lemma 3.3.1 implies that

$$\|\boldsymbol{Q}(k+N)\, \boldsymbol{Q}(k+N-1)\cdots \boldsymbol{Q}(k)\|_\infty \leq \gamma < 1, \tag{3.27}$$

for some γ [48]. This implies that

$$\max_{[0,T]} \left\|\Delta \boldsymbol{n}^{k+N}(t)\right\|_\infty \leq \gamma \max_{[0,T]} \left\|\Delta \boldsymbol{n}^{k}(t)\right\|_\infty \tag{3.28}$$

which proves the theorem. □

The monotonicity conditions on the electric field are approximately satisfied in practice on the horizontal lines in the channel between the source and drain of a MOSFET. It is encouraging that a sup-norm contractivity result can be proven at all. This is in sharp contrast to the behavior of WR when used for the circuit simulation of typical MOSFET circuits with feedback, in which sup-norm contractivity is rare [50].

4

The WORDS Program

4.1 Introduction

Both standard and waveform methods have been implemented in the device transient simulation program WORDS [33]. This chapter gives an overview of the WORDS program. The chapter begins with a brief description of some of the implementation issues that arise when WR is applied to device simulation. Finally, in Section 4.3, experimental results are given, comparing standard direct solution methods to waveform methods. In particular, the results show that block WR accelerated with WRN and timestep refinement is competitive on a serial machine with pointwise direct methods.

4.2 Implementation: the WORDS Program

The need for both efficiency and accuracy affects all aspects of the design and implementation of the WORDS program, as described in the following sections. WORDS consists of about 15,000 lines of C.

4.2.1 Vertical-Line Block WR

In practice, the node-by-node Gauss-Jacobi WR Algorithm 3.1.1 may require many hundreds or thousands of WR iterations to converge, severely limiting the efficiency of WR-based device simulation. To reduce the number of WR iterations to achieve convergence, WORDS uses the block relaxation techniques described in Section 2.2.4. The block WR algorithm for devices is similar to the node-by-node WR Algorithm 3.1.1, except that the 3×3 systems of equations for each node are replaced by fewer, larger

systems of equations representing the equations of all the nodes contained in each block.

Although using blocks of nodes accelerates device WR convergence, the computational expense of directly solving the larger system of equations of a block of nodes is higher than that of solving the smaller 3×3 system of equations for a single node. In addition, the same timepoints must be used in the waveforms of all nodes within a block, so that a block algorithm cannot take advantage of any multirate behavior *within* a block. The challenge is to use a blocking scheme that covers the device mesh in relatively few, easy-to-solve blocks and groups tightly-coupled nodes together, but does not group nodes which are expected to change at different rates.

The blocking scheme used in WORDS divides the tensor-product mesh into vertical line blocks. This is a particularly effective blocking strategy for MOSFET simulation for the following reasons:

1. The vertical line blocks directly capture the behavior of the oxide-silicon interface. Computing the interface behavior is a numerically difficult problem because of the Neumann reflecting boundary condition on the electron and hole current equations.

2. Since each vertical line is essentially a one-dimensional device simulation problem, the resulting subsystems produce 3×3-block-tridiagonal matrices which are easily solved in linear time.

3. The potentials at either end of a vertical line in the channel of a MOSFET are pinned by the gate and substrate contacts, so that each line's solution correctly accounts for these contacts and directly captures the highly nonlinear electric field dependence governing surface depletion and channel width from the very first WR iteration.

4. Vertical line blocking allows the WR algorithm to take advantage of the multirate behavior resulting from the horizontal distance between the source and drain contacts. For example, if the drain potential of a MOSFET is increased while the source potential is held fixed, more timepoints are needed to accurately resolve the widening of the drain depletion region than are needed to resolve the source end of the device.

5. The vertical line blocks can be processed in red/black order to maintain parallelizability when Gauss-Seidel WR is used [44].

4.2.2 Waveform Relaxation Newton Acceleration

When block WR is applied to the nonlinear differential-algebraic system of equations of a device, the differential-algebraic systems of equations solved for each block at each timepoint are nonlinear, and require multiple Newton iterations. As described in Section 2.4.2, the waveform relaxation Newton (WRN) method reduces the computation per WR iteration and improves WR efficiency [50, 37]. When WRN is applied to the nonlinear differential-algebraic system of equations of a device, the nonlinear differential-algebraic systems of equations of each block are linearized about the solution of the previous iteration, and the linearized systems are solved with a single matrix solution at each timepoint.

4.2.3 Integration Method

To solve the differential-algebraic system of each block generated by spatially discretizing the device equations, WORDS uses the backward Euler method and the variable-timestep second-order backward differentiation formula (BDF2) [11, 5]. For the first two timesteps (and also for the first two timesteps after any other discontinuity in the derivative of a contact voltage), WORDS uses the backward Euler method, to avoid having to look backward in time past a discontinuity. Thereafter, WORDS uses the second-order backward difference formula.

4.2.4 Timestep Selection Strategy for Multirate WR

Since WORDS can take advantage of multirate behavior within a device by using different timepoints for different blocks of nodes, some interpolation in time is unavoidable. Before computing a block's solution at a time t, the solution of the adjacent blocks at time t must be obtained from the previous WR iteration's waveforms, in order to compute the fluxes and currents along mesh edges to the adjacent blocks. Furthermore, if the WRN method is to be used, the block's previous solution is required as an initial guess at time t. In either case, if t is not a timepoint used in the waveforms of the previous WR iteration, then interpolation in time is necessary. WORDS uses polynomial interpolation of the same order as the integration method used to compute the first waveform timepoint greater than t.

To minimize both the number of interpolations and their associated errors, WORDS synchronizes adjacent block timesteps by selecting all timesteps of all blocks before each WR iteration. This *a priori* timestep control strategy also results in efficient timepoint

ALGORITHM 4.2.1 (WRN WITH TIMESTEP HALVING STRATEGY).

Initialize: Choose a global, minimal set of timepoints $\{t_*\}$ for all subsystems.

For $k = 0, 1, \ldots$ until converged

For each subsystem i

If $k = 0$ then let $\{t_i^{k+1}\} = \{t_*\}$.

Otherwise,

let $\{t_i^{k+1}\} = \{t_i^k\}$.

For each $t_i^k(j)$, add $\frac{1}{2}\left(t_i^k(j-1) + t_i^k(j)\right)$ to $\{t_i^{k+1}\}$ if

1. the LTE in waveform $\boldsymbol{x}_i^k$ at timestep $t_i^k(j)$ is too large, or
2. neighboring subsystems use much smaller timesteps.

Use $\{t_i^{k+1}\}$ to solve for waveforms $\boldsymbol{x}_i^{k+1}$ of subsystem i.

placement, algorithmic parallelism, and the possibility of refinement with iteration [37]. On the first WR iteration, each block of nodes is solved on a globally-shared, minimal set of timepoints consisting of the times of any slope discontinuities of the input voltages on the device contacts. Before each subsequent WRN iteration, the timesteps for a block are chosen by computing a standard predictor-corrector local truncation error (LTE) estimate [5] for each non-fixed timestep of the block's previous iteration electron and hole concentration waveforms. If the LTE of a timestep made in the previous iteration's waveform is larger than a tolerance computed from the sup-norm of the waveform, then a new timepoint is introduced into the waveform to precisely halve the old timestep. The timestep selection strategy is shown in Algorithm 4.2.1, with $\{t_i^k(1), t_i^k(2), \ldots\}$ denoting the set of timepoints with which the waveforms of subsystem i at iteration k are computed.

The primary advantage of this timestep-halving strategy is that adjacent blocks will tend to select similar sets of timepoints, since all blocks begin with the same timepoints and select new ones by halving intervals. The synchronization of adjacent blocks is fostered by also requiring a block to split a timestep if the corresponding timestep was twice split by the neighboring blocks. In addition, except for the newly-added timepoints, a block uses the same set of timepoints as it did in the previous WR iteration. This synchronization with previous iterations reduces interpolation of the initial guess for WRN. Because the necessity of interpolation for adjacent block solutions and previous

iteration solutions is reduced, the timestep-halving strategy improves the accuracy of the solution and terminal current calculation, while maintaining most of the multirate computational advantage of WR.

4.2.5 Terminal Current Calculation

In WORDS, each device contact is represented as a connected group of nodes with a single, externally-specified potential waveform. In the following, the mesh edges between the nodes i of a contact and the adjacent non-contact mesh nodes j will be referred to as *contact edges*. The contacts and the contact edges of the MOSFET device shown in Figure 1-1 are shown in Figure 4-1.

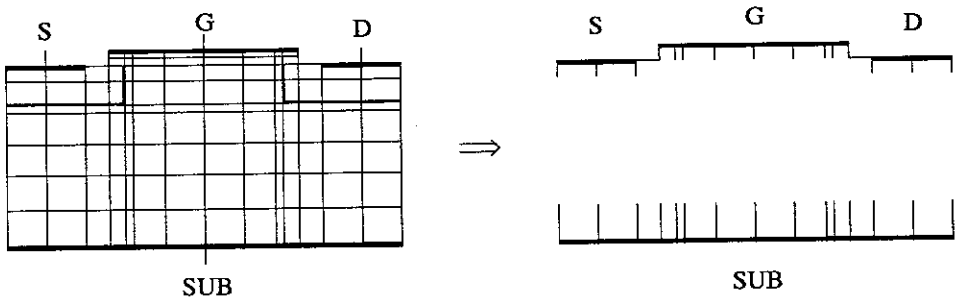

FIGURE 4-1: The contacts and contact edges of a MOSFET device.

The total current I entering a contact from within a device is the sum of the electron, hole and displacement currents travelling on its contact edges,

$$I = \sum_{i,j} d_{ij} \left[J_{n_{ij}} + J_{p_{ij}} + J_{q_{ij}} \right] \tag{4.1}$$

where for each contact edge between nodes i and j, d_{ij} is the length of the side of the Voronoi box that encloses node i and bisects the edge. The electron current flux $J_{n_{ij}}$ and the hole current flux $J_{p_{ij}}$ are computed from the u, n and p solutions at both ends of each contact edge,

$$J_{n_{ij}} = \frac{kT}{q} \frac{\mu_{n_{ij}}}{L_{ij}} \left[n_i B(u_i - u_j) - n_j B(u_j - u_i) \right] \tag{4.2}$$

$$J_{p_{ij}} = \frac{kT}{q} \frac{\mu_{p_{ij}}}{L_{ij}} \left[p_i B(u_j - u_i) - p_j B(u_i - u_j) \right]. \tag{4.3}$$

To compute the displacement current on a contact edge, first the charge on the contact edge is computed by using the Poisson equation to relate the charge to the electric field on the edge,

$$Q_{ij} = \frac{kT}{q} \frac{\epsilon_{ij}}{L_{ij}} \left[u_i - u_j \right].$$

The displacement current flux $J_{p_{ij}}$ is the time derivative of Q_{ij}, computed with the same multistep integration method coefficients used to compute $\frac{d}{dt}$ of n and p.

Because WORDS may use different timepoints at different nodes to take advantage of multirate behavior, some care must be taken to avoid the introduction of interpolation error when computing a terminal current I. In the highly-doped source and drain regions, any interpolation error, particularly of potential, may produce non-negligible errors in terminal current waveforms.

In WORDS, a current waveform is computed and associated with each contact edge, representing the behavior in time of the total current travelling on that edge. Each current waveform is constructed using whatever timepoints are used by the u_j, n_j and p_j waveforms of the contact edge's non-contact endpoint j. This choice of timepoints allows the contact edge current waveforms to be computed without interpolation, because the potential waveform of a contact is externally-specified, and the electron and hole concentrations are constant. WORDS uses these contact edge current waveforms for internal calculations such as convergence checking and timestep refinement.

To compute the terminal currents of a device, i.e. the summations of the current waveforms on the contact edges of each contact, WORDS uses the *union* of all the timepoints used to compute the edge current waveform of the contact. Here, with multirate behavior, some interpolation in time is unavoidable, but is minimized by the synchronization of the adjacent block timepoints discussed in Section 4.2.4.

4.2.6 Convergence Testing

The primary convergence criterion used in WORDS is a requirement that the maximum error of the contact edge current waveforms be less than some tolerance. The maximum error is estimated from the change and rate of change of contact edge currents between successive iterations [13].

Let $I_j^k(t)$ denote the total current travelling on the jth contact edge at time $t \in [0,T]$ computed after WR iteration k, and let ΔI^k denote the maximum over all contact edges of the device of the sup-norm of the change of contact edge current from iteration $k-1$ to k, i.e.

$$\Delta I^k = \max_j \left\| I_j^k(t) - I_j^{k-1}(t) \right\|_\infty \tag{4.4}$$

and let $\rho = \Delta I^k / \Delta I^{k-1}$ denote the convergence rate. Assuming linear convergence, an estimate of the distance between the exact solution I^* and the kth iteration solution I^k is

$$I^* - I^k = \rho \Delta I^k + \rho^2 \Delta I^k + \cdots = \left(\frac{\rho}{1-\rho} \right) \Delta I^k$$

Therefore, in WORDS, the WR iterations are terminated when

$$\left(\frac{\rho}{1-\rho}\right)\Delta I^k \leq r \cdot \max_j \left\|I_j^k(t)\right\|_\infty + a$$

where the maximum is taken over all contact edges of the device, r is a relative tolerance (typically 10^{-3}) and a is some small absolute tolerance (typically 10^{-6} A/cm)

4.2.7 Convergence-Driven Timestep Refinement

WORDS uses a convergence-driven timestep refinement strategy that can reduce the computation required in each WRN iteration, thereby improving WRN efficiency [37]. As with the ordinary timestep-halving strategy in Section 4.2.4, the timesteps of all blocks are chosen before every iteration. All block waveforms begin with a global, minimal set of timepoints, and a timestep is halved if the standard LTE estimate computed for the previous iteration waveform is larger than the acceptable LTE tolerance.

With a timestep refinement strategy, the acceptable LTE tolerance is not a constant, but is a factor that is decreased slowly with iteration, thereby adding timepoints to waveforms to keep their estimated local truncation errors less than some fraction of the error criterion used to measure convergence. To implement convergence-driven refinement in WORDS, the constant acceptable LTE tolerance used in the ordinary timestep-halving strategy is multiplied by the factor

$$\gamma = \max\left(1, \frac{\Delta I^k}{\left(r \cdot \max_j \left\|I_j^k(t)\right\|_\infty + a\right)}\right),$$

where ΔI^k and the denominator are the contact edge current values used to determine convergence, as described above in Section 4.2.6. In the early WRN iterations, the factor γ is greater than one, and the LTE tolerance is "looser" than in the ordinary timestep-halving strategy. As iterations proceed and ΔI^k decreases, the acceptable LTE tolerance approaches the value used in the non-refinement timestep strategy.

Note that the factor γ is computed from the unextrapolated difference (4.4) and not the estimated error $I^* - I^k$. This ensures that the LTE is kept tight enough to maintain accuracy in the ΔI waveforms, producing smooth edge current waveforms. Although this does not prevent timepoints from being added early in the iterations of a slowly converging problem, this does prevent spurious spikes from occurring in the edge current waveforms as new timepoints are added.

4.3 Experimental Results

In this section we present results from experiments with the WORDS program. Simulation examples were constructed from the three different n-channel MOS devices described in table 4-1. Each device was spatially discretized on an irregular tensor-product mesh, i.e. the mesh lines were placed closer together at points where u, n and p were expected to exhibit rapid spatial variation.

device	description	L	L_{eff}	t_{ox}	mesh	unknowns
ldd	lightly-doped drain	0.64	0.4	19	15×20	656
soi	silicon-on-insulator	0.7	0.5	7	18×24	856
kar	abrupt junction	3.0	2.2	50	19×31	1379

Table 4-1: Description of MOS devices: gate length L (μm), effective channel length L_{eff} (μm), oxide thickness t_{ox} (nm), mesh size (rows$\times$cols), and total number of u, n and p unknowns. Silicon thickness of soi is $t_{si} = 0.1$ μm.

The three MOS devices were used to construct the six simulation examples described in table 4-2. Dirichlet boundary conditions were imposed by a gate contact and by ohmic contacts at the drain, the source, and along the bottom of the substrate. Neumann reflecting boundary conditions were imposed along the left and right edges of the meshes. For all examples, the source and substrate contacts were fixed at 0 V. The drain-driven karD test is illustrated in Figure 4-2. Note that all six of the examples involve both high and low current conditions.

example	device	drain bias	gate bias
lddD	ldd	5 psec, 0-5 V ramp	5 V
lddG	ldd	5 V	5 psec, 0-5 V ramp
soiD	soi	5 psec, 0-5 V ramp	5 V
soiG	soi	5 V	5 psec, 0-5V ramp
karD	kar	50 psec, 0-5 V ramp	5 V
karG	kar	5 V	50 psec, 0-5 V ramp

Table 4-2: Applied bias conditions in the simulation examples.

The WORDS program is written in C, and all experiments were run on an IBM RS/6000 model 540 workstation.

4.3.1 Comparisons to Direct Solution

To compare the computational efficiency of the WR methods to standard direct solution, WORDS was used to compute the transient behavior of each of the six examples.

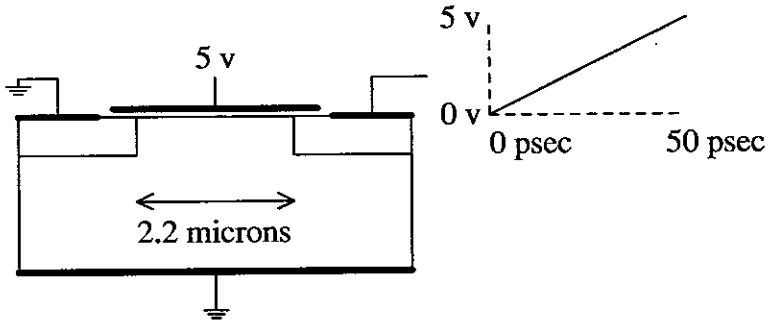

FIGURE 4-2: The drain-driven karD example.

Table 4-3 shows the CPU times required by direct solution, Gauss-Seidel WR and WRN with convergence driven timestep refinement. The timing results show that WRN with timestep refinement is competitive with direct methods.

example	direct	WR	(iters)	WRN/r
lddD	147.47	594.74	(353)	296.48
lddG	439.22	615.04	(268)	253.19
soiD	208.77	244.12	(235)	96.68
soiG	130.75	226.66	(161)	95.61
karD	697.08	1536.34	(315)	501.66
karG	2622.29	1104.08	(215)	563.75

Table 4-3: CPU times for direct solution, the WR method and WRN with timestep refinement, simulating a drain ramp and a gate ramp applied to the three devices.

As described in previous sections, the WR methods were implemented as block iterative methods, i.e. the device meshes were broken into blocks defined by its vertical lines and the equations governing nodes in the same block were solved simultaneously using the first and second-order backward differentiation formulae, Newton's method and sparse Gaussian elimination. The blocks were processed in red/black order to indicate parallelizability. To compute the solution directly, each device mesh was treated as a single block.

4.3.2 Terminal Current Accuracy

For all of the simulations of table 4-3, the terminal currents, including displacement currents, were computed accurately to one part in 10^3. Figure 4-3 compares the terminal current accuracy of direct solution and the WORDS method for the karG example.

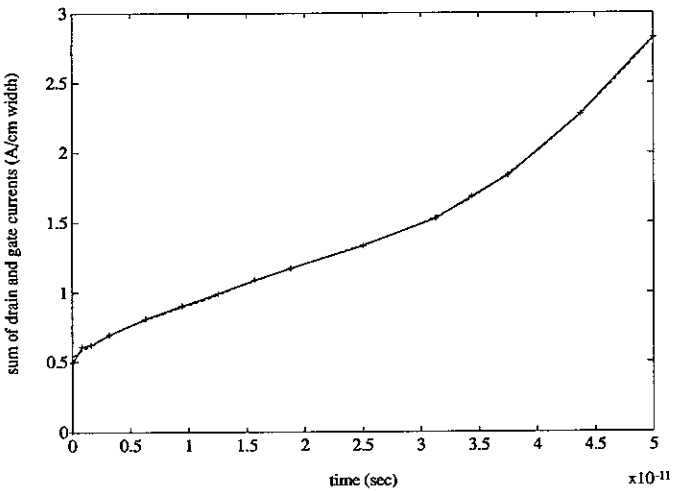

FIGURE 4-3: Terminal current accuracy comparison between a waveform method and direct methods, showing the sum of resistive and displacement currents into the drain and gate terminals, in response to the 50 psec gate ramp from 0 to 5 volts in the karG test.

4.3.3 Multirate Behavior

When accelerated with WRN and convergence-driven timestep refinement, the device WR method is sometimes faster than direct solution because WR takes advantage of multirate behavior. Figure 4-4 illustrates the number of timepoints required per vertical line block to solve the karD example with WR. Different blocks required different numbers of timesteps because more timepoints were needed to resolve the widening of the drain depletion region than were needed to resolve the source end of the device. It is interesting to note that different nodes changed at different rates, not so much because the electron and hole concentrations changed at different times, but because they changed by different orders of magnitude (i.e. *multimagnitude* behavior).

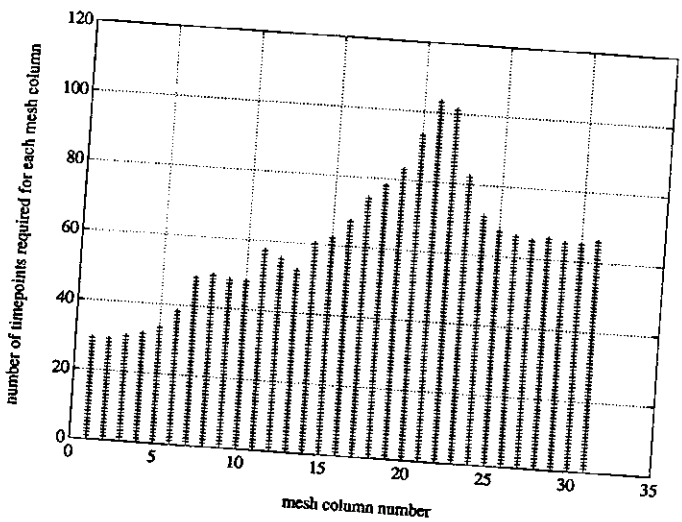

FIGURE 4-4: Multirate behavior in the karD WR test.

5

Convolution Successive Overrelaxation

5.1 Introduction

To achieve highest performance on a parallel computer, a numerical method must avoid frequent parallel synchronization [1]. The waveform relaxation approach to solving time-dependent initial-value problems is just such a method, as the iterates are vector waveforms over an interval, rather than vectors at single timepoints [24, 50, 18]. Like any relaxation scheme, efficiency depends on rapid convergence, and there have been several investigations into how to accelerate WR [24, 42], including using multigrid [19] and conjugate direction techniques [20].

In this chapter, we investigate using successive overrelaxation (SOR) to accelerate WR convergence. In particular, we show that the pessimistic results about dynamic SOR iteration derived in [24] can be substantially improved by replacing multiplication with a fixed SOR parameter by convolution with an SOR kernel. We derive the optimal SOR kernel using Fourier analysis and z-transform techniques and demonstrate the effectiveness of the approach for a model parabolic problem. Finally, we demonstrate the general applicability of the approach, by using the method to solve the time-dependent drift-diffusion equations associated with modeling semiconductor devices.

We begin in Section 5.2 by reviewing waveform SOR, and in Section 5.3 we relate the algorithm to pointwise SOR, to demonstrate the difficulty in accelerating WR with a fixed SOR parameter. In Section 5.4, we use Fourier analysis to informally derive the optimal convolution SOR kernel for the continuous WR algorithm. In Section 5.5, we consider the effect of time-discretization, we derive the optimal convolution SOR sequence, and give a

proof of optimality. In Section 5.6, we briefly describe some aspects of implementing the convolution SOR algorithm. In Section 5.7 we apply the method to device simulation on a serial machine.

5.2 Waveform SOR

In this section, we consider applying waveform relaxation methods to the model linear initial-value problem

$$\left(\tfrac{d}{dt} + \boldsymbol{A}\right) \boldsymbol{x}(t) = \boldsymbol{b}(t) \quad \text{with} \quad \boldsymbol{x}(0) = \boldsymbol{x}_0, \tag{5.1}$$

where $\boldsymbol{A} \in \mathbb{R}^{n\times n}$, $\boldsymbol{b}(t) \in \mathbb{R}^n$ is given for all $t \in [0,T]$, and $\boldsymbol{x}(t) \in \mathbb{R}^n$ is to be computed.

Consider the standard relaxation splitting $\boldsymbol{A} = \boldsymbol{D}-\boldsymbol{L}-\boldsymbol{U}$, where $\boldsymbol{D}$, $\boldsymbol{L}$ and $\boldsymbol{U}$ are the diagonal, strictly lower triangular and strictly upper triangular pieces of $\boldsymbol{A}$ [48, 52]. Subtracting successive waveform relaxation iterations, the waveform Gauss-Jacobi (WGJ) and waveform Gauss-Seidel (WGS) iteration equations, respectively, may be written as:

$$\left(\tfrac{d}{dt} + \boldsymbol{D}\right) \Delta\boldsymbol{x}^{k+1}(t) = (\boldsymbol{L}+\boldsymbol{U})\, \Delta\boldsymbol{x}^{k}(t) \tag{5.2}$$

$$\left(\tfrac{d}{dt} + \boldsymbol{D} - \boldsymbol{L}\right) \Delta\boldsymbol{x}^{k+1}(t) = \boldsymbol{U}\, \Delta\boldsymbol{x}^{k}(t), \tag{5.3}$$

where $\Delta\boldsymbol{x}^{k+1}(t) = \boldsymbol{x}^{k+1}(t) - \boldsymbol{x}^{k}(t)$.

The waveform SOR method for acceleration of WGS is a simple extension of algebraic SOR. To derive the waveform SOR iteration equation, compute a waveform $\hat{x}_i^{k+1}(t)$ on $t \in [0,T]$, solving an initial value problem as in WGS:

$$\begin{aligned} \left(\tfrac{d}{dt} + a_{ii}\right) \hat{x}_i^{k+1}(t) &= b_i(t) - \sum_{j=1}^{i-1} a_{ij} x_i^{k+1}(t) - \sum_{j=i+1}^{n} a_{ij} x_i^{k}(t) \\ \hat{x}_i^{k+1}(0) &= x_{0_i}, \end{aligned} \tag{5.4}$$

and then update $x_i^k(t)$ in the iteration direction by multiplication with an overrelaxation parameter ω,

$$x_i^{k+1}(t) \leftarrow x_i^k(t) + \omega \cdot \left[\hat{x}_i^{k+1}(t) - x_i^k(t)\right]. \tag{5.5}$$

Combining equations (5.4) and (5.5) yields

$$\begin{aligned} &\left(\tfrac{d}{dt} + a_{ii}\right) x_i^{k+1}(t) = \\ &\quad (1-\omega)\left[\left(\tfrac{d}{dt} + a_{ii}\right) x_i^{k}(t)\right] + \omega \left[b_i(t) - \sum_{j=1}^{i-1} a_{ij} x_i^{k+1}(t) - \sum_{j=i+1}^{n} a_{ij} x_i^{k}(t)\right], \end{aligned}$$

which, after subtracting successive waveform relaxation iterations, leads to

$$(\tfrac{d}{dt} + \boldsymbol{D} - \omega\boldsymbol{L})\,\Delta\boldsymbol{x}^{k+1}(t) = [(1-\omega)(\tfrac{d}{dt} + \boldsymbol{D}) + \omega\boldsymbol{U}]\,\Delta\boldsymbol{x}^{k}(t), \tag{5.6}$$

where $\Delta\boldsymbol{x}^{k+1}(t) = \boldsymbol{x}^{k+1}(t) - \boldsymbol{x}^{k}(t)$.

Note that deleting the $\frac{d}{dt}$ terms in equations (5.2), (5.3) and (5.6) results in precisely the standard algebraic relaxation and SOR iteration equations. Also note that waveform SOR as defined by (5.6) is *not* precisely the same as the dynamic SOR iteration considered in [24].

In this paper, we will consider the effect of replacing the overrelaxation multiplication in (5.5) by an *overrelaxation convolution* with a time-dependent SOR kernel $\omega(t)$. In the continuous time case, the overrelaxation equation becomes

$$x_i^{k+1}(t) \leftarrow x_i^{k}(t) + \int_{-\infty}^{\infty} \omega(\tau) \cdot \left[\hat{x}_i^{k+1}(t-\tau) - x_i^{k}(t-\tau)\right] d\tau, \tag{5.7}$$

and the iteration equation for the resulting method is

$$\begin{aligned}(\tfrac{d}{dt} + \boldsymbol{D})\,\Delta\boldsymbol{x}^{k+1}(t) - \boldsymbol{L}\int_{-\infty}^{\infty} \omega(\tau)\,\Delta\boldsymbol{x}^{k+1}(t-\tau)\,d\tau = \\ (\tfrac{d}{dt} + \boldsymbol{D})\,\Delta\boldsymbol{x}^{k}(t) + [\boldsymbol{U} - (\tfrac{d}{dt} + \boldsymbol{D})]\int_{-\infty}^{\infty} \omega(\tau)\,\Delta\boldsymbol{x}^{k}(t-\tau)\,d\tau.\end{aligned} \tag{5.8}$$

5.3 Relation to Pointwise SOR

Discretizing (5.1) in time using a multistep integration method [11] yields

$$\begin{aligned}\left[\boldsymbol{I} + h\beta_0\boldsymbol{A}\right]\boldsymbol{x}[m] = \\ h\beta_0\boldsymbol{b}[m] + \sum_{j=1}^{s}\left\{h\beta_j\left(\boldsymbol{b}[m-j] - \boldsymbol{A}\boldsymbol{x}[m-j]\right) - \alpha_j\boldsymbol{x}[m-j]\right\},\end{aligned} \tag{5.9}$$

where α_j and β_j are the coefficients of the multistep method, and $\boldsymbol{x}[m]$ denotes the discrete approximation to $\boldsymbol{x}(t)$ at $t = mh$. We now compare the convergence rate of the waveform SOR method to the convergence rate of pointwise SOR, in which algebraic SOR is used to solve the matrix problem at each timepoint.

The pointwise SOR iteration equations are derived by applying the relaxation splitting $\boldsymbol{A} = \boldsymbol{D} - \boldsymbol{L} - \boldsymbol{U}$ to equation (5.9) and taking the difference between the $(k+1)$st and kth iterations. More precisely, the pointwise SOR iteration equation applied to solve (5.9) for $\Delta\boldsymbol{x}^{k+1}[m] = \boldsymbol{x}^{k+1}[m] - \boldsymbol{x}^{k}[m]$ is

$$\begin{aligned}\left[(\boldsymbol{I} + h\beta_0\boldsymbol{D}) - \omega h\beta_0\boldsymbol{L}\right]\Delta\boldsymbol{x}^{k+1}[m] = \\ \left[(1-\omega)(\boldsymbol{I} + h\beta_0\boldsymbol{D}) + \omega h\beta_0\boldsymbol{U}\right]\Delta\boldsymbol{x}^{k}[m],\end{aligned} \tag{5.10}$$

where ω is the SOR parameter. It follows that the spectral radius of the iteration matrix generated by pointwise SOR at the mth timestep is

$$\rho\left(\left[(\boldsymbol{I}+h\beta_0\boldsymbol{D})-\omega h\beta_0\boldsymbol{L}\right]^{-1}\left[(1-\omega)(\boldsymbol{I}+h\beta_0\boldsymbol{D})+\omega h\beta_0\boldsymbol{U}\right]\right). \tag{5.11}$$

If waveform SOR is used to solve the model problem (5.1), and a multistep method is used to solve iteration equation (5.6), then $\Delta\boldsymbol{x}^{k+1}[m]$ satisfies

$$\begin{aligned}\sum_{j=0}^{s}\alpha_j\left[\Delta\boldsymbol{x}^{k+1}[m-j]-(1-\omega)\,\Delta\boldsymbol{x}^{k}[m-j]\right]=\\ h\sum_{j=0}^{s}\beta_j\Big\{-(\boldsymbol{D}-\omega\boldsymbol{L})\,\Delta\boldsymbol{x}^{k+1}[m-j]+[(1-\omega)\,\boldsymbol{D}+\omega\boldsymbol{U}]\,\Delta\boldsymbol{x}^{k}[m-j]\Big\}.\end{aligned} \tag{5.12}$$

This can be rewritten as the discrete-time analogue of (5.6):

$$\begin{aligned}\sum_{j=0}^{s}\left[(\alpha_j\boldsymbol{I}+h\beta_j\boldsymbol{D})-\omega h\beta_j\boldsymbol{L}\right]\Delta\boldsymbol{x}^{k+1}[m-j]=\\ \sum_{j=0}^{s}\left[(1-\omega)(\alpha_j\boldsymbol{I}+h\beta_j\boldsymbol{D})+\omega h\beta_j\boldsymbol{U}\right]\Delta\boldsymbol{x}^{k}[m-j].\end{aligned} \tag{5.13}$$

As the similarities of equations (5.10) and (5.13) suggest, if the time interval is finite, i.e. the number of timesteps is some finite L, then for a given timestep h and a given SOR parameter ω, the time-discretized waveform SOR method has the same asymptotic convergence rate as the pointwise SOR method.

THEOREM 5.3.1. *On a finite simulation interval, the iterations defined by (5.10) and (5.13) have the same asymptotic convergence rate.*

Proof. Let $\boldsymbol{y}^k$ denote the large vector consisting of the concatenation of vectors $\Delta\boldsymbol{x}^k[m]$ at all L discrete timepoints, i.e. $\boldsymbol{y}^k=\left[\Delta\boldsymbol{x}^k[1]^T,\ldots,\Delta\boldsymbol{x}^k[L]^T\right]^T$. Collecting together the equations (5.13) generated at each timepoint into one large matrix equation in terms of vectors $\boldsymbol{y}^{k+1}$ and $\boldsymbol{y}^k$ yields $\boldsymbol{M}\Delta\boldsymbol{y}^{k+1}=\boldsymbol{N}\Delta\boldsymbol{y}^k$ where $\boldsymbol{M},\boldsymbol{N}\in\mathbb{R}^{Ln\times Ln}$ are block lower triangular banded matrices, with blocks of size $n\times n$, and with block bandwidth s. It is then easily seen that $\boldsymbol{M}^{-1}\boldsymbol{N}$ is block lower triangular, with diagonal blocks equal to

$$\left[(\boldsymbol{I}+h\beta_0\boldsymbol{D})-\omega h\beta_0\boldsymbol{L}\right]^{-1}\left[(1-\omega)(\boldsymbol{I}+h\beta_0\boldsymbol{D})+\omega h\beta_0\boldsymbol{U}\right]. \tag{5.14}$$

Therefore, $\rho(\boldsymbol{M}^{-1}\boldsymbol{N})$ is given by (5.11), implying that the iterations defined by (5.10) and (5.13) have identical asymptotic convergence rates. □

Theorem 5.3.1 suggests that parameter ω for waveform SOR should be chosen to be precisely equal to the optimum parameter for the pointwise SOR method. However, this does not necessarily lead to fastest convergence, as the following example illustrates.

EXAMPLE 5.3.1. Let $t \in [0, 2048]$, $\boldsymbol{x}(0) = 0$, and let matrix $\boldsymbol{A} \in \mathbb{R}^{32\times 32}$ and time-dependent input vector $\boldsymbol{b}(t) \in \mathbb{R}^{32}$ of the model problem (5.1) be given by

$$\begin{aligned} \boldsymbol{A} &= \begin{bmatrix} 2 & -1 & & \\ -1 & 2 & \ddots & \\ & \ddots & \ddots & -1 \\ & & -1 & 2 \end{bmatrix} \\ \boldsymbol{b}(t) &= \begin{bmatrix} b_1(t) \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad \text{where} \quad b_1(t) = \begin{cases} 1 - \cos\left(\dfrac{2\pi t}{256}\right) & \text{if } t \leq 256 \\ 0 & \text{otherwise.} \end{cases} \end{aligned} \tag{5.15}$$

Consider the four problems generated by discretizing in time with the first-order backward difference formula, using 64, 128, 256, and 512 uniform timesteps of size $h = 32$, 16, 8 and 4 respectively.

Since the tridiagonal matrix $\boldsymbol{A}$ is symmetric and is consistently ordered [52, 48], the matrix $(\boldsymbol{I} + h\beta_0\boldsymbol{A})$ of the pointwise time-discretized model problem (5.9), is also consistently ordered, and the optimum pointwise SOR parameter ω_{opt} is given by

$$\omega_{opt} = \frac{2}{1 + \sqrt{1 - \mu_1^2}} \tag{5.16}$$

where $\mu_1 = \rho(\boldsymbol{H}_{GJ})$ is the spectral radius of the pointwise Gauss-Jacobi iteration matrix $\boldsymbol{H}_{GJ} = (\boldsymbol{I} + h\beta_0\boldsymbol{D})^{-1}(h\beta_0\boldsymbol{L} + h\beta_0\boldsymbol{U})$. For the four problems with 64, 128, 256 and 512 timesteps, the optimum pointwise parameters ω_{opt} are approximately 1.669, 1.586, 1.482 and 1.364 respectively.

Curves PT64, PT128, PT256 and PT512 of Figure 5-1 show convergence versus iteration of the waveform SOR method for the four problems with their optimum pointwise SOR parameters ω_{opt}. Note that as the total number of timesteps is increased, the initial convergence rate slows, approaching a limiting value of the convergence rate of the unaccelerated Gauss-Seidel WR algorithm (shown as WR in Figure 5-1). In each case, the convergence rate of waveform SOR eventually approaches the expected asymptotic value of $\omega_{opt} - 1$. Note that with a reasonable error accuracy tolerance such as 10^{-6}

as a stopping point, the asymptotic convergence rate is *never* reached. For comparison, Figure 5-1 also shows the superposition of four convergence plots of the new convolution SOR method (CSOR) to be introduced in the following sections. The four CSOR runs have the same convergence rate, independent of the number of timesteps.

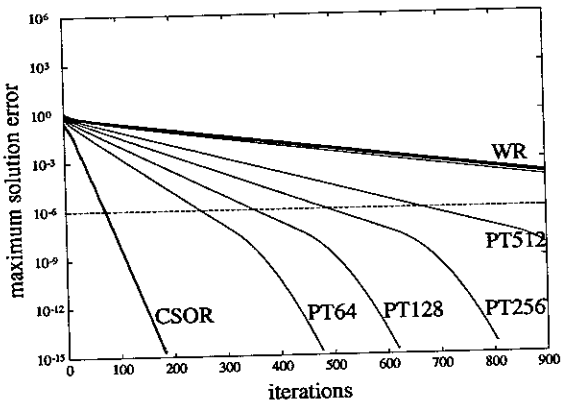

FIGURE 5-1: Convergence of waveform SOR using the pointwise optimal parameter (PT) compared to waveform relaxation (WR), and waveform convolution SOR (CSOR), with 64, 128, 256 and 512 timesteps. Note that the four CSOR runs have the same convergence rate.

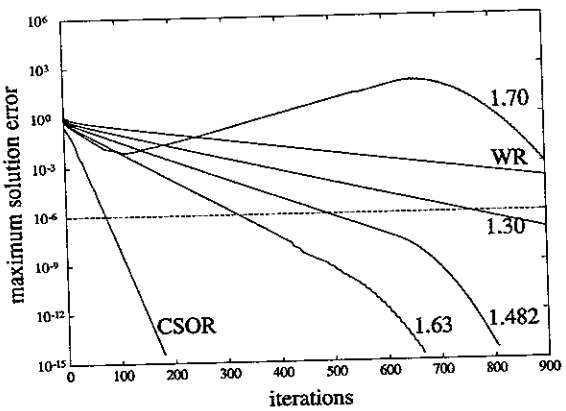

FIGURE 5-2: Effect on convergence of the 256-timestep waveform SOR of varying the SOR parameter from the pointwise optimum $\omega_{opt} = 1.482$.

To illustrate the effect of choosing a different SOR parameter ω, Figure 5-2 shows the convergence versus iteration of the 256-timestep example for waveform SOR with values of the SOR parameter ω not equal to the pointwise optimum $\omega_{opt} = 1.482$. When $\omega = 1.30 < \omega_{opt}$, the convergence curve lies between the pointwise optimum curve and the WR convergence curve, i.e. both initial and asymptotic convergence rates are slower.

By increasing the SOR parameter to $\omega = 1.63 > \omega_{opt}$, the initial convergence rate can be made faster at the expense of slowing down the asymptotic convergence rate. But as the $\omega = 1.70$ curve shows, once the SOR parameter is increased beyond some point, the waveform SOR method may appear to diverge before eventually converging. The intermediate solutions produced by the $\omega = 1.70$ example contain spurious oscillations, as shown in Figure 5-3. Note both the growth and translation of the oscillation with iteration.

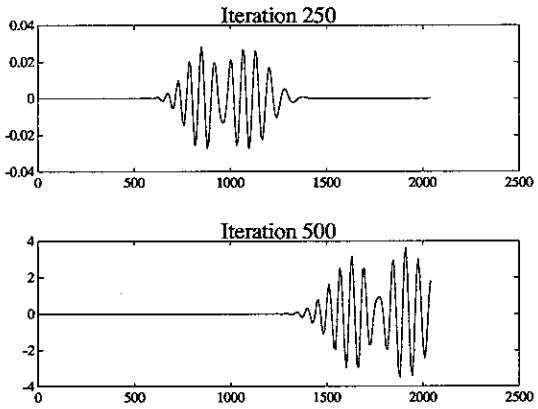

FIGURE 5-3: Delta waveform $\Delta x_{16}^{k+1}(t) = x_{16}^{k+1}(t) - x_{16}^{k}(t)$ versus time after iterations 250 and 500, for the 256-timestep waveform SOR method using $\omega = 1.70$, showing the growth and translation of an oscillating region. Note that the vertical scales on the axes differ by two orders of magnitude.

In general, the optimum pointwise SOR parameter ω_{opt} does not dramatically improve the convergence rate of waveform SOR because the matrix $\boldsymbol{M}^{-1}\boldsymbol{N}$ which describes the waveform SOR convergence is far from normal. Note that this is the case even if $\boldsymbol{D}^{-1}(\boldsymbol{L}+\boldsymbol{U})$ is normal. This suggests that although the spectral radius of the iteration matrix determines the *asymptotic* convergence rate of waveform SOR, it does not determine the effective observable convergence rate. The effective convergence rate could be characterized, for example, by computing the pseudo-eigenvalues of the waveform SOR iteration matrix [47]. In the following section, we take an alternate approach.

5.4 Informal Fourier Analysis and Convolution SOR

In [24], the spectral radius of dynamic iteration operators

$$\left(\tfrac{d}{dt} + \boldsymbol{M}\right) \Delta \boldsymbol{x}^{k+1} = \boldsymbol{N} \Delta \boldsymbol{x}^{k}, \tag{5.17}$$

where $\boldsymbol{M}, \boldsymbol{N} \in \mathbb{R}^{n \times n}$, such as WGJ (5.2) or WGS (5.3), was related to their Fourier transform. In this section, we make a more detailed, but less formal use of Fourier analysis to derive a convolution SOR kernel for the waveform SOR operator of equation (5.6).

The Fourier transform of a function $\boldsymbol{y}(t)$ is given by

$$\boldsymbol{y}(i\Omega) \equiv \int_{-\infty}^{\infty} \boldsymbol{y}(t)\, e^{-i\Omega}\, dt \equiv \mathcal{F}\, \boldsymbol{y}(t), \tag{5.18}$$

where Ω is frequency. Assuming that $\Delta \boldsymbol{x}^k$ is infinitely differentiable, and that the Fourier transform $\Delta \boldsymbol{x}^k(i\Omega) = \mathcal{F}\, \Delta \boldsymbol{x}^k(t)$ exists, standard Fourier identities can be used to derive the iteration equation $\Delta \boldsymbol{x}^{k+1}(i\Omega) = \boldsymbol{H}(i\Omega)\, \Delta \boldsymbol{x}^k(i\Omega)$, where for WGJ, WGS and waveform SOR, the iteration operator $\boldsymbol{H}(i\Omega)$ is given by

$$\boldsymbol{H}_{GJ}(i\Omega) = \boldsymbol{D}_\Omega^{-1}(\boldsymbol{L} + \boldsymbol{U}) \tag{5.19}$$

$$\boldsymbol{H}_{GS}(i\Omega) = (\boldsymbol{D}_\Omega - \boldsymbol{L})^{-1}\boldsymbol{U} \tag{5.20}$$

$$\boldsymbol{H}_{SOR}(i\Omega) = (\boldsymbol{D}_\Omega - \omega\boldsymbol{L})^{-1}[(1-\omega)\boldsymbol{D}_\Omega + \omega\boldsymbol{U}], \tag{5.21}$$

respectively, where $\boldsymbol{D}_\Omega = i\Omega\boldsymbol{I} + \boldsymbol{D}$. An informal interpretation of equations (5.19)–(5.21) is that the spectral radius $\rho(\boldsymbol{H}(i\Omega))$ yields the asymptotic convergence rate for errors in the frequency component Ω.

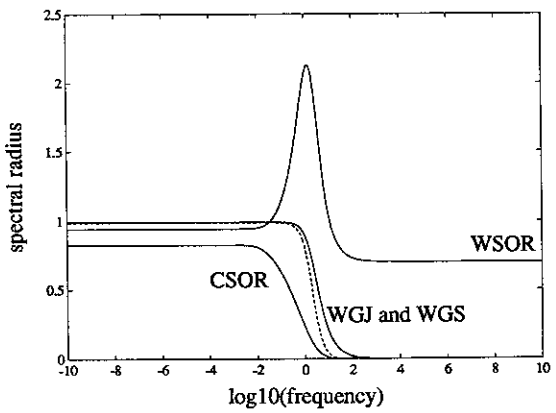

FIGURE 5-4: The spectral radii as functions of frequency Ω of the Gauss-Jacobi WR, Gauss-Seidel WR (dashed) and waveform SOR iteration matrices for the 32×32 version of the continuous-time problem of Example 5.3.1, with $\omega = 1.70$ for waveform SOR. For comparison, the plot also shows the spectral radius versus frequency for the convolution SOR method using the optimal convolution SOR sequence.

Figure 5-4 is a plot of the spectral radii of $\boldsymbol{H}_{GJ}(i\Omega)$, $\boldsymbol{H}_{GS}(i\Omega)$ and $\boldsymbol{H}_{SOR}(i\Omega)$ for the 32×32 continuous-time problem given in Example 5.3.1, using $\omega = 1.70$ for $\boldsymbol{H}_{SOR}(i\Omega)$. For comparison, the plot also shows the spectral radius versus frequency for the convolution SOR method (dash-dot) using the optimal convolution SOR sequence. For the

32×32 problem, the WGJ and WGS spectral radius is only slightly less than unity for low frequencies, and high frequency components of the error are damped much more quickly than low frequency components. However, the spectral radius $\rho(\boldsymbol{H}_{SOR}(i\Omega))$ is greater than unity over a range of frequencies, and therefore the waveform SOR iteration magnifies errors in this frequency range. This effect is similar to the behavior predicted in [24] and is easily seen in Figures 5-2 and 5-3.

The situation can be remedied by using a generalized SOR algorithm, *convolution SOR* (CSOR), in which equation (5.5) is replaced by overrelaxation convolution (5.7). The Fourier transform of the convolution equation (5.7) is

$$x_i^{k+1}(i\Omega) \leftarrow x_i^k(i\Omega) + \omega(i\Omega) \cdot \left[\hat{x}_i^{k+1}(i\Omega) - x_i^k(i\Omega)\right], \tag{5.22}$$

where $\omega(i\Omega)$ is the Fourier transform of the time-dependent $\omega(t)$. Thus, in effect, the CSOR method uses a different SOR parameter for each frequency. The Fourier transform of the resulting CSOR operator is given by

$$\boldsymbol{H}_C(i\Omega) = \left[\boldsymbol{D}_\Omega - \omega(i\Omega)\,\boldsymbol{L}\right]^{-1}\left[\left(1-\omega(i\Omega)\right)\boldsymbol{D}_\Omega + \omega(i\Omega)\,\boldsymbol{U}\right], \tag{5.23}$$

where $\boldsymbol{D}_\Omega = i\Omega\boldsymbol{I} + \boldsymbol{D}$, as before.

5.5 Discrete-Time Analysis

The analysis of the convolution SOR method can be made more precise with the aid of the z-transform. For a sequence $\boldsymbol{y}[m]$, the unilateral z-transform, $\boldsymbol{y}(z) = \mathcal{Z}\,\boldsymbol{y}[m]$, is defined by

$$\boldsymbol{y}(z) \equiv \sum_{m=0}^{\infty} \boldsymbol{y}[m]\, z^{-m} \equiv \mathcal{Z}\,\boldsymbol{y}[m], \tag{5.24}$$

where $z \in \mathbb{C}$ [27]. Using the transform representation, the WGJ, WGS and ordinary waveform SOR methods applied to the time-discretized problem (5.9) are of the form $\Delta\boldsymbol{x}^{k+1}(z) = \boldsymbol{H}(z)\,\Delta\boldsymbol{x}^k(z)$, where

$$\boldsymbol{H}_{GJ}(z) = \boldsymbol{D}_z^{-1}(\boldsymbol{L}+\boldsymbol{U}) \tag{5.25}$$

$$\boldsymbol{H}_{GS}(z) = (\boldsymbol{D}_z - \boldsymbol{L})^{-1}\boldsymbol{U} \tag{5.26}$$

$$\boldsymbol{H}_{SOR}(z) = (\boldsymbol{D}_z - \omega\boldsymbol{L})^{-1}[(1-\omega)\boldsymbol{D}_z + \omega\boldsymbol{U}]. \tag{5.27}$$

For a general multistep method with uniform timestep h, the diagonal matrix $\boldsymbol{D}_z$ is

$$\boldsymbol{D}_z = \frac{\sum_{j=0}^{s}\alpha_j z^{-j}}{h\sum_{j=0}^{s}\beta_j z^{-j}}\,\boldsymbol{I} + \boldsymbol{D}. \tag{5.28}$$

Like equations (5.19)–(5.21), these operators correspond to standard relaxation and SOR matrices with $\boldsymbol{D}$ replaced by the diagonal matrix $\boldsymbol{D}_z$. Note that $\boldsymbol{H}_{SOR}(z)$ in (5.27) can be obtained by applying the z-transform to (5.13).

To derive the iteration equation for discrete-time convolution SOR, let overrelaxation equation (5.5) be replaced by a convolution sum with sequence $\omega[m]$,

$$x_i^{k+1}[m] \leftarrow x_i^k[m] + \sum_{\ell=0}^{m} \omega[\ell] \cdot \left(\hat{x}_i^{k+1}[m-\ell] - x_i^k[m-\ell]\right). \tag{5.29}$$

The z-transform of this convolution equation is

$$x_i^{k+1}(z) \leftarrow x_i^k(z) + \omega(z) \cdot \left[\hat{x}_i^{k+1}(z) - x_i^k(z)\right], \tag{5.30}$$

where $\omega(z)$ is the z-transform of the sequence $\omega[m]$. The z-transform of the resulting discrete-time CSOR operator is given by

$$\boldsymbol{H}_C(z) = \left[\boldsymbol{D}_z - \omega(z)\,\boldsymbol{L}\right]^{-1}\left[\left(1-\omega(z)\right)\boldsymbol{D}_z + \omega(z)\,\boldsymbol{U}\right], \tag{5.31}$$

with $\omega(z) = \mathcal{Z}\,\omega[m]$ and $\boldsymbol{D}_z$ is given by (5.28) above.

The CSOR operator on sequences is derived by combining the convolution equation (5.29) with the discrete-time problem (5.9) and subtracting successive iterations. The resulting CSOR iteration equation for a general multistep method is given by

$$\begin{aligned}
&\sum_{j=0}^{s} (\alpha_j \boldsymbol{I} + h\beta_j \boldsymbol{D})\, \Delta\boldsymbol{x}^{k+1}[m-j] \\
&\quad - \sum_{j=0}^{s} h\beta_j \boldsymbol{L} \sum_{\ell=0}^{m-j} \omega[\ell]\, \Delta\boldsymbol{x}^{k+1}[m-j-\ell] \quad = \\
&\sum_{j=0}^{s} (\alpha_j \boldsymbol{I} + h\beta_j \boldsymbol{D})\, \Delta\boldsymbol{x}^{k}[m-j] \\
&\quad + \sum_{j=0}^{s} \left[h\beta_j \boldsymbol{U} - (\alpha_j \boldsymbol{I} + h\beta_j \boldsymbol{D})\right] \sum_{\ell=0}^{m-j} \omega[\ell]\, \Delta\boldsymbol{x}^{k}[m-j-\ell]
\end{aligned} \tag{5.32}$$

Though lengthy, this is precisely the same as the waveform SOR iteration equation (5.13), with overrelaxation multiplication replaced by a convolution sum.

Let $\mathcal{K}$ denote the CSOR operator mapping from sequence $\Delta\boldsymbol{x}^k[m]$ to $\Delta\boldsymbol{x}^{k+1}[m]$, defined by

$$\Delta\boldsymbol{x}^{k+1}[m] = \mathcal{K}\Delta\boldsymbol{x}^k[m] \equiv \sum_{\ell=0}^{m} \boldsymbol{h}_C[\ell]\, \Delta\boldsymbol{x}^k[m], \tag{5.33}$$

where $\boldsymbol{h}_C[m] \in \mathbb{R}^{n\times n}$ is derived from equation (5.32). Concatenating the vectors $\Delta\boldsymbol{x}[m]$ at all time points, the operator $\mathcal{K}$ may be written in block matrix form as

$$\begin{bmatrix} \boldsymbol{M}_1 & & & \\ \boldsymbol{M}_2 & \boldsymbol{M}_1 & & \\ \boldsymbol{M}_3 & \boldsymbol{M}_2 & \boldsymbol{M}_1 & \\ \vdots & \ddots & \ddots & \ddots \end{bmatrix} \begin{bmatrix} \Delta\boldsymbol{x}^{k+1}[1] \\ \Delta\boldsymbol{x}^{k+1}[2] \\ \Delta\boldsymbol{x}^{k+1}[3] \\ \vdots \end{bmatrix} = \begin{bmatrix} \boldsymbol{N}_1 & & & \\ \boldsymbol{N}_2 & \boldsymbol{N}_1 & & \\ \boldsymbol{N}_3 & \boldsymbol{N}_2 & \boldsymbol{N}_1 & \\ \vdots & \ddots & \ddots & \ddots \end{bmatrix} \begin{bmatrix} \Delta\boldsymbol{x}^{k}[1] \\ \Delta\boldsymbol{x}^{k}[2] \\ \Delta\boldsymbol{x}^{k}[3] \\ \vdots \end{bmatrix} \tag{5.34}$$

From this, it is easily seen that the CSOR operator $\mathcal{K}$ is a block Toeplitz operator [30], corresponding to the semi-infinite, block lower triangular matrix $\boldsymbol{M}^{-1}\boldsymbol{N}$ specified by (5.32). This leads to the following lemma.

LEMMA 5.5.1. *On the infinite interval, the spectral radius of the operator $\mathcal{K}$ defined in (5.33) is determined by the spectral radius of the matrix $\boldsymbol{H}_C(z)$ given by (5.31),*

$$\rho(\mathcal{K}) = \max_{|z|\leq 1} \rho(\boldsymbol{H}_C(z)). \tag{5.35}$$

Proof. Because operator $\mathcal{K}$ is a block lower triangular semi-infinite Toeplitz operator, the spectrum of $\mathcal{K}$ equals the spectrum of the matrix-valued symbol $f(z)$ of the Toeplitz operator, where z ranges over the closed unit disk [30]. The symbol of the Toeplitz operator $\mathcal{K}$ is precisely the matrix $\boldsymbol{H}_C(z)$. □

Thus, minimizing the spectral radius of the CSOR operator on sequences is the same as minimizing the spectral radius of the matrix $\boldsymbol{H}_C(z)$ given by (5.31). This leads to the following theorem, the main result of this section.

THEOREM 5.5.2. *If, at a particular $z \in \mathbb{C}$, the spectrum $\mu(z)$ of $\boldsymbol{H}_{GJ}(z)$ lies on a line segment $[-\mu_1(z), \mu_1(z)]$, with $\mu_1(z) \in \mathbb{C}$ and $|\mu_1(z)| < 1$, then the spectral radius of $\boldsymbol{H}_C(z)$ is minimized by the unique optimum $\omega_{opt}(z) \in \mathbb{C}$ given by*

$$\omega_{opt}(z) = \frac{2}{1 + \sqrt{1 - \mu_1(z)^2}} \tag{5.36}$$

where $\sqrt{\cdot}$ denotes the root with the positive real part. Furthermore, the sequence $\omega_{opt}[m] = \mathcal{Z}^{-1}\omega_{opt}(z)$ is optimal in the sense that it minimizes the spectral radius of the operator $\mathcal{K}$.

Proof. For brevity, the argument (z) will be omitted in the following, i.e. ω will denote $\omega(z)$ and $\boldsymbol{H}_C$ will denote the convolution SOR operator (evaluated at z) computed using CSOR parameter $\omega(z)$.

Let $\mu_i = r_i\mu_1$, where $r_i \in [-1,1]$, denote each eigenvalue of $\boldsymbol{H}_{GJ}$ given by (5.25). Classical SOR theory [52, 48] guarantees that for each $\mu_i = r_i\mu_1$, there is an eigenvalue λ_i of $\boldsymbol{H}_C$ which satisfies

$$\lambda_i - \omega r_i \mu_1 \sqrt{\lambda_i} + (\omega - 1) = 0, \tag{5.37}$$

and therefore, from the quadratic formula,

$$\sqrt{\lambda_i} = \frac{r_i\mu_1\omega}{2} + \sqrt{\left(\frac{r_i\mu_1\omega}{2}\right)^2 - \omega + 1}. \tag{5.38}$$

Let ω be the conjectured optimal ω_{opt}. Combining equation (5.36) with (5.38) yields

$$\sqrt{|\lambda_i|} = \left|\tfrac{1}{2}\mu_1\omega_{opt}\left[r_i + \sqrt{r_i^2 - 1}\right]\right| = \left|\tfrac{1}{2}\mu_1\omega_{opt}\right|, \tag{5.39}$$

where the rightmost equality follows from the fact that $\left|r_i + \sqrt{r_i^2 - 1}\right| = 1$ for $r_i \in [-1,1]$. And as (5.39) holds for all i, with parameter $\omega = \omega_{opt}$,

$$\rho(\boldsymbol{H}_C) = |\lambda_i| = \left|\left(\tfrac{1}{2}\mu_1\omega_{opt}\right)^2\right| = |\omega_{opt} - 1|. \tag{5.40}$$

Equation (5.40) implies that $\rho(\boldsymbol{H}_C)$ cannot be decreased by picking a value of ω such that $|\omega - 1| > |\omega_{opt} - 1|$. This follows from the fact that,

$$\rho(\boldsymbol{H}_C) \geq |\omega - 1| \tag{5.41}$$

for any ω [52, 48].

To show that $\rho(\boldsymbol{H}_C)$ also cannot be decreased by choosing a value of ω such that $|\omega - 1| < |\omega_{opt} - 1|$, consider the eigenvalue λ_j corresponding to μ_1:

$$\sqrt{\lambda_j} = f_+(\omega) = \frac{\mu_1\omega}{2} + \sqrt{\frac{\mu_1^2\omega^2}{4} - \omega + 1} \tag{5.42}$$

and note that $f_+ : \mathbb{C} \to \mathbb{C}$, given by equation (5.42), is a single-valued, continuous function that is analytic except at

$$\omega_1, \omega_2 = \frac{2}{1 \pm \sqrt{1 - \mu_1^2}}. \tag{5.43}$$

Since $|\mu_1| < 1$, points ω_1 and ω_2 lie in the interior and exterior, respectively, of the circle $|\omega - 1| = 1$ in the complex ω-plane. Note that ω_1 equals the conjectured ω_{opt} from equation (5.36).

Let D denote the interior of the curve given by the perimeter of the circle $|\omega - 1| = 1$ with a cut along the line defined by the circle's center and ω_1. The cut follows the line

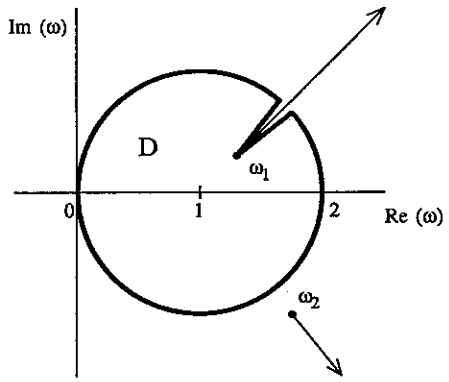

FIGURE 5-5: The region D and branch cuts in the complex ω-plane.

from the perimeter down to ω_1, and then back up the other side to the perimeter, as shown in Figure 5-5. The function f_+ is nonzero everywhere within D, since equation (5.37) implies that a zero can occur only at $\omega = 1$, and $f_+(1) = \mu_1$. Therefore, the minimum modulus theorem [36] implies that $|f_+(\omega)|$ attains its minimum value somewhere on the boundary of D. Finally, the lower bound in (5.41) implies that $\omega_1 = \omega_{opt}$ in (5.36) is the only point on D which can achieve as low a $\rho(\boldsymbol{H}_C)$ as given in (5.40), completing the proof. □

Note that when the eigenvalues μ lie on a real line segment, this theorem contains the classic SOR Theorem [52, 48] as a special case, and extends the theorem to complex matrices [8, 52]. Whereas the classic SOR theorem requires the spectrum μ of the Gauss-Jacobi matrix $\boldsymbol{H}_{GJ}$ to lie on the real line segment $[-\mu_1, \mu_1]$ with $|\mu_1| < 1$, the CSOR theorem requires the spectrum $\mu(z)$ of $\boldsymbol{H}_{GJ}(z)$ to lie on a line segment $[-\mu_1(z), \mu_1(z)]$, with $\mu_1(z) \in \mathbb{C}$ and $|\mu_1(z)| < 1$, as shown in Figure 5-6. For the discrete-time problem (5.9), this requirement is satisfied, for example, by the class of diagonally-dominant symmetric matrices $\boldsymbol{A}$ with constant diagonal $\boldsymbol{D} = d\boldsymbol{I}$, since the spectrum $\mu(z)$ of the $\boldsymbol{H}_{GJ}(z)$ for such matrices is

$$\mu(z) = \frac{\mu_0 dh \sum_{j=0}^{s} \beta_j z^{-j}}{\sum_{j=0}^{s} [\alpha_j z^{-j} + dh\beta_j z^{-j}]} \tag{5.44}$$

where μ_0 denotes the eigenvalues of $\boldsymbol{D}^{-1}(\boldsymbol{L} + \boldsymbol{U})$. For any particular z, the real line segment μ_0 is mapped to a rotated line segment.

Theorem 5.5.2 leads immediately to the following corollary.

COROLLARY 5.5.3. *If $\omega_{opt}(z)$ given by (5.36) is analytic, then the corresponding sequence $\omega_{opt}[m] = \mathcal{Z}^{-1}\ \omega_{opt}(z)$ is optimal for discrete-time convolution SOR, i.e. it*

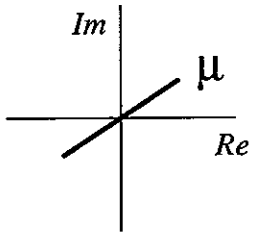

FIGURE 5-6: The optimal CSOR parameter theorem requires the spectrum $\mu(z)$ to lie on a line segment $[-\mu_1(z), \mu_1(z)]$, with $\mu_1(z) \in \mathbb{C}$ and $|\mu_1(z)| < 1$.

minimizes spectral radius of the operator on sequences, $\rho(\mathcal{K})$. In addition, the following is true.

1. *Sequence $\omega_{opt}[m] = \mathcal{Z}^{-1}\, \omega_{opt}(z)$ is real.*

2. *Initial value $\omega_{opt}[0]$ equals ω_{opt} of pointwise SOR.*

3. *Asymptotic convergence on a finite interval equals that of optimal pointwise SOR.*

Proof. Since $\omega(z)$ minimizes the spectral radius of $\boldsymbol{H}_C(z)$ for any z, Lemma 5.5.1 proves that convolution with the inverse z-transform $\omega_{opt}[m] = \mathcal{Z}^{-1}\, \omega_{opt}(z)$ optimally minimizes the spectral radius of operator $\mathcal{K}$.

1. The diagonal matrix $\boldsymbol{D}_z$ given by (5.28) is a conjugate-symmetric function of z, so that $\boldsymbol{H}_{GJ}(z)$ given by (5.25), and its largest-magnitude eigenvalue $\mu_1(z)$ are also conjugate-symmetric functions of z. This implies that $\omega_{opt}(z)$ is also conjugate-symmetric, and the inverse z-transform is real.

2. The initial value theorem for the z-transform [14] implies that

$$\omega_{opt}[0] = \lim_{z\to\infty} \omega_{opt}(z) = \frac{2}{1 + \sqrt{1 - \left(\lim_{z\to\infty} \mu_1(z)\right)^2}}.$$

   In the limit $z \to \infty$, the matrix $\boldsymbol{H}_{GJ}(z)$ reduces precisely to the pointwise Gauss-Jacobi iteration matrix used to solve (5.9), since

$$\lim_{z\to\infty} \boldsymbol{D}_z^{-1}(\boldsymbol{L} + \boldsymbol{U}) = \left(\frac{1}{h\beta_0}\, \boldsymbol{I} + \boldsymbol{D}\right)^{-1} (\boldsymbol{L} + \boldsymbol{U}).$$

3. Referring to the block matrix expression of CSOR operator $\mathcal{K}$ (5.34), it is easily seen that on a finite simulation interval, the spectrum of the block lower triangular operator $\mathcal{K}$ is equal to the spectrum of its block diagonal $\boldsymbol{M}_1^{-1}\boldsymbol{N}_1$ where

$$\begin{aligned}\boldsymbol{M}_1 &= [(\boldsymbol{I}+h\boldsymbol{D})-\omega_{opt}[0]h\boldsymbol{L}] \\ \boldsymbol{N}_1 &= [(1-\omega_{opt}[0])(\boldsymbol{I}+h\boldsymbol{D})+\omega_{opt}[0]h\boldsymbol{U}].\end{aligned}$$

Since $\omega[0]$ equals the optimal ω_{opt} of pointwise SOR, the matrix $\boldsymbol{M}_1^{-1}\boldsymbol{N}_1$ is precisely equal to the iteration matrix of optimal pointwise SOR.

□

Note that the result that the asymptotic convergence of optimal convolution SOR on a finite interval equals that of optimal pointwise SOR is nearly irrelevant, since the operator $\mathcal{K}$ is far from normal. As Example 5.3.1 and Figure 5-1 showed for waveform SOR, the asymptotic convergence rate given by the spectral radius may have little to do with the convergence rate observed in practice.

5.6 Implementation of Convolution SOR

In practice, standard SOR algorithms use the results of several Gauss-Jacobi iterations to estimate the optimal SOR parameter [13], but it is not yet clear how to apply this approach to convolution SOR. One might first try to estimate the largest-magnitude eigenvalue $\mu_1(z)$, by computing the transforms of successive WR iterates $\Delta\boldsymbol{x}^{k+1}[m]$. Then an adaptive scheme might be used to approximate $\omega_{opt}[m]$. Unfortunately, it is the complex value of $\mu_1(z)$ that is required, not the magnitude $|\mu_1(z)|$. Furthermore, this approach immediately runs into difficulties, since the iterates are not known for m greater than some finite L, so it is not possible, in general, to compute the transform of the iterates. A more successful approach has been to consider the spectrum of the SOR operator, and use a power method to estimate $\mu_1(z)$ and $\omega_{opt}[m]$.

For the Poisson problem in Example 5.3.1 and Figures 5-1 and 5-2, the optimum convolution SOR sequence $\omega_{opt}[m]$ was computed analytically. First, the largest-magnitude eigenvalue $\mu_1 = \rho(\boldsymbol{D}^{-1}(\boldsymbol{L}+\boldsymbol{U}))$ for the Poisson matrix was calculated. Theorem 5.5.2 implies that the z-transform of the optimal CSOR sequence is given by

$$\omega_{opt}(z) = \frac{2}{1+\sqrt{1-\left(\dfrac{\mu_1 dh}{1-z^{-1}+dh}\right)^2}}. \tag{5.45}$$

ALGORITHM 5.6.1 (APPROXIMATION OF $\omega_{opt}[m]$).

1. *Compute* $\mu_1(z) = \rho(\boldsymbol{H}_{GJ}(z))$ for several values of z (e.g. $z = 1, -1, \infty, \ldots$).
2. *Compute* the corresponding values of $\omega_{opt}(z) = \dfrac{2}{1 + \sqrt{1 - \mu_1(z)^2}}$.
3. *Fit* a low-order rational function to the values of $\omega_{opt}(z)$.
4. *Compute* inverse z-transform $\omega_{opt}[m] = \mathcal{Z}^{-1}\, \omega_{opt}(z)$.

Finally, the inverse z-transform of $\omega_{opt}(z)$ was computed analytically by series expansion.

For the device experiments, the CSOR sequence $\omega_{opt}[m]$ was computed before beginning any WR iterations. First, power iterations were used to estimate the largest-magnitude eigenvalue $\mu_1(z)$ at several specific values of z (e.g. $z = 1, -1, \infty, \ldots$). Note that for real values of z, this simply amounts to adding $(1 - z^{-1})/h$ to the diagonal elements of $\boldsymbol{A}$ and computing the algebraic Gauss-Jacobi spectral radius. These values of $\mu_1(z)$ were used in formula (5.36) to compute values of $\omega_{opt}(z)$. Next, the computed values of $\omega_{opt}(z)$ are fitted by a ratio of low-order polynomials in z^{-1}:

$$\omega_{opt}(z) \approx \frac{\sum_{k=0}^{M} b_k z^{-k}}{\sum_{k=0}^{N} a_k z^{-k}} = \sum_{k=0}^{N} \frac{c_k}{1 - r_k z^{-1}}. \tag{5.46}$$

Finally, the inverse z-transform is applied, yielding

$$\omega_{opt}[m] \approx \sum_{k=0}^{N} c_k r_k^m.$$

Note that because the resulting $\omega_{opt}[m]$ is a simple sum of exponentials in time, the computational expense of the overrelaxation convolution is reduced to that of only a few multiplications and accumulations at each time point. A summary of the computation of $\omega_{opt}[m]$ is given in Algorithm 5.6.1.

Of course, the CSOR theory does not directly apply to the device simulation problem, primarily because the system is nonlinear. And even when the time-discretized problem is linearized about some point, the linearization is time-dependent, and the resulting matrices do not necessarily satisfy the conditions of the optimal CSOR parameter theorem. Nevertheless, as Section 5.7 will show, the CSOR inherits some of the robustness of the SOR method, and can successfully be applied to the device transient simulation problem.

5.7 Experimental Results

Various solution methods have been implemented in the WR-based device transient simulation program WORDS [33]. This section contains experimental results with WORDS comparing the performance of the WRN method with and without convolution SOR acceleration to that of standard pointwise methods.

As described in Chapter 4, the waveform methods in WORDS are based on red/black block Gauss-Seidel WR, where the blocks correspond to vertical mesh lines. The sequence of implicit nonlinear algebraic systems generated for each block by the backward difference formula are solved with Newton's method, and the linear equation systems generated by Newton's method are solved with sparse Gaussian elimination. The particularly efficient WRN algorithm is derived by taking only one Newton iteration at each timestep, using an initial guess obtained from the previous waveform iterate at each timestep. To provide an initial guess for WRN and for WRN with convolution SOR acceleration, 16 or 32 initial WR iterations were used. For convolution SOR acceleration, the "optimal" CSOR sequence $\omega[m]$ was determined by linearizing the device equations (1.9)–(1.11) about the initial condition solution, and fitting $\omega_{opt}(z)$ with a rational function as described in Section 5.6. To diminish the effect of the nonlinearity, the overrelaxation convolution was applied only to the potential variables u.

For the pointwise methods, the device equation system of the entire mesh was generated successively at each time point, with each block contributing the corresponding "block row" of equations. The nonlinear algebraic system resulting from the backward difference formula was solved at each time point with Newton's method, and the linear equation system generated by Newton's method was solved with either sparse Gaussian elimination or the block Gauss-Jacobi preconditioned GMRES conjugate direction algorithm [35]. To simplify comparisons to the waveform methods, the same vertical-line blocking scheme was used for the GMRES algorithm.

device	description	L_{eff}	t_{ox}	mesh	unknowns
ldd	lightly-doped drain	0.4	19	15×20	656
soi	silicon-on-insulator	0.5	7	18×24	856
				18×64	2292
kar	abrupt junction	2.2	50	19×31	1379
				19×64	2854

Table 5-1: Description of MOS devices.

The three MOS devices of Table 5-1 were used to construct six simulation examples, each device being subjected to either a drain voltage pulse with the gate held high (the D

examples), or a gate voltage pulse with the drain held high (the G examples). To observe the effect on WR convergence, two additional gate-pulsed examples were constructed by refining the device meshes of karG and soiG to contain 64 vertical lines. All eight examples ranged from low to high drain current, and in the G examples, the gate displacement current was substantial because the applied voltage pulses changed at a rate of $.2 \sim 2$ volts per picosecond. Dirichlet boundary conditions were also imposed by ohmic contacts at the source, and along the bottom of the substrate, both held at zero volts. Neumann reflecting boundary conditions were imposed along the left and right edges of the meshes. The drain-driven karD test setup is illustrated in Figure 5-7.

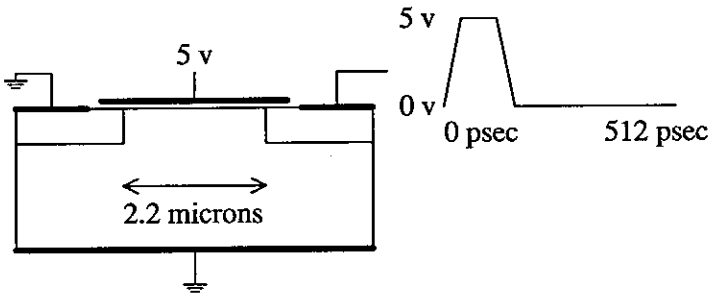

FIGURE 5-7: Illustration of the drain-driven karD example.

To simplify comparisons between the different solution methods, the backward Euler method using 256 equal timesteps was used for all experiments, on a simulation interval of 51.2 or 512 picoseconds. The convergence criterion for all experiments and all methods was the requirement that the maximum error over the simulation interval in the value of any terminal current be less than one part in 10^4. Note that using global uniform timesteps eliminates the ability of WR to exploit multirate behavior, one of the primary computational advantages of WR on a serial machine. Nevertheless, as the results will show, even without this multirate advantage, the accelerated WR algorithms are competitive on a serial machine with the best pointwise methods.

Figure 5-8 shows the convergence of the eight examples as a function of iteration for the ordinary waveform relaxation Newton method (WRN) and the same method accelerated with convolution SOR. Despite the nonlinearity of the semiconductor equations, the convolution SOR algorithm converged substantially faster than the WRN method, demonstrating the robustness of the approach.

Table 5-2 shows the CPU times in seconds required for solution of the eight examples, for pointwise methods and waveform methods. The serial experiments were performed on an IBM RS/6000 Model 540 workstation, with 256 Meg of memory. For the pointwise methods, the *sparse elim* column shows the result of using direct factorization to solve the

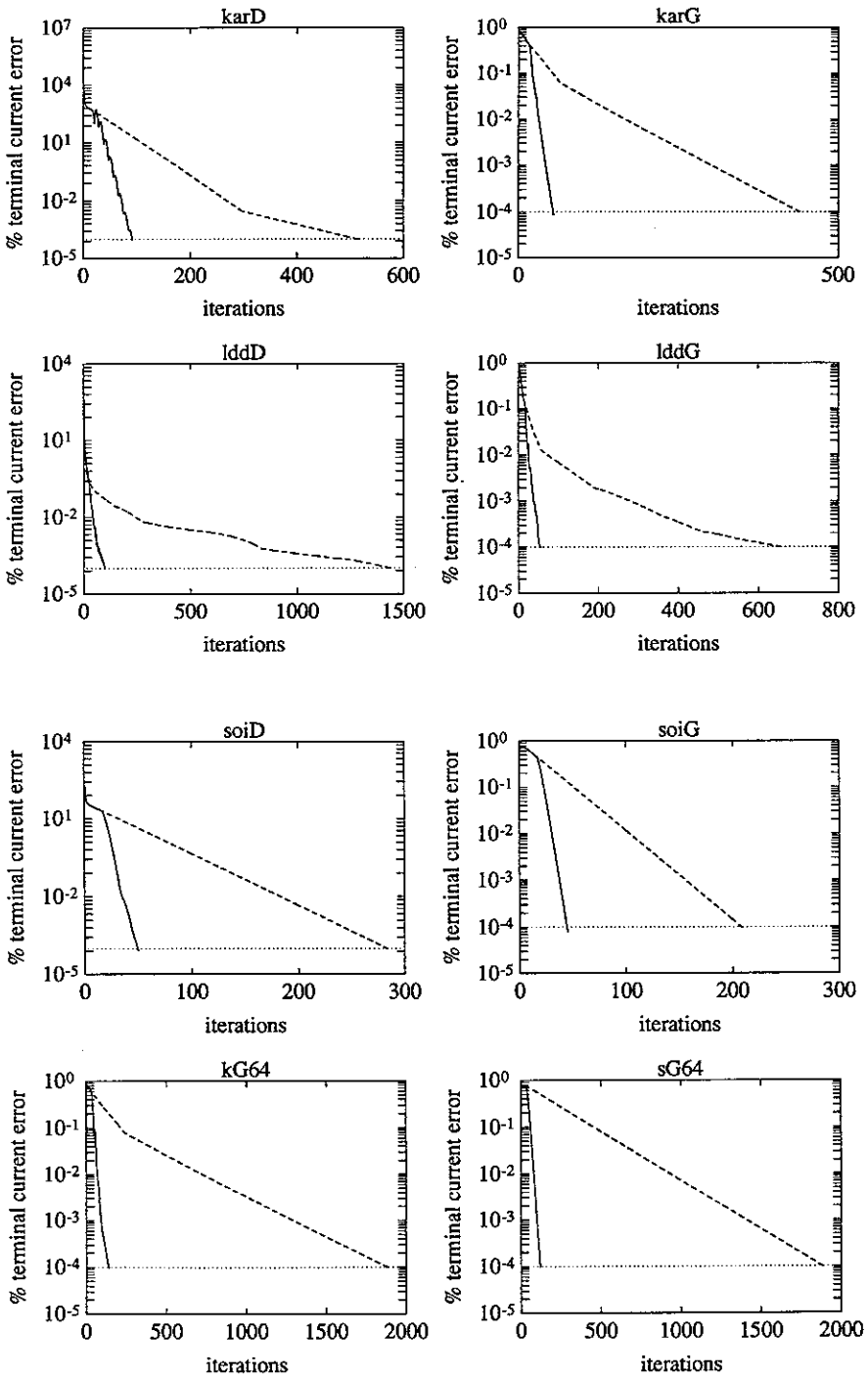

FIGURE 5-8: Terminal current error versus iteration, for WR (dashed) and waveform convolution SOR (solid).

	Pointwise Methods			Waveform Methods			
example	sparse elim	GMRES	(iters)	WRN	(iters)	conv SOR	(iters)
lddD	231.65	620.00	(53741)	7327.24	(1434)	584.81	(100)
lddG	244.46	505.92	(43680)	3404.19	(657)	360.15	(55)
soiD	401.95	188.53	(12363)	1832.21	(284)	388.62	(50)
soiG	430.80	205.28	(13466)	1377.21	(209)	369.16	(45)
karD	1401.60	673.81	(26022)	5451.02	(515)	1088.41	(92)
karG	1460.38	794.01	(30858)	4672.49	(440)	712.74	(55)
soiG64	2144.05	2249.01	(48176)	31399.19	(1885)	2239.39	(119)
karG64	3833.04	5205.89	(71316)	40246.82	(1885)	3328.85	(141)

Table 5-2: Comparison of serial times required for pointwise methods and waveform methods. Serial experiments were conducted on an IBM RS/6000 Model 540 workstation.

matrix problem at each time point, compared to using the iterative algorithm GMRES. The waveform method columns show the result of using ordinary Gauss-Seidel WRN, and the same algorithm accelerated with convolution SOR. The results show that on a serial machine, convolution SOR is competitive with pointwise Newton-GMRES – in fact, convolution SOR is slightly faster than pointwise Newton-GMRES for 5 of the 8 examples. This is especially encouraging since the fixed timestep simulations do not allow the waveform method to take advantage of multirate behavior.

6

Parallel WR for Device Simulation

6.1 Introduction

As shown in the previous chapter, the convolution SOR method is competitive with the best known pointwise method on a serial machine. In this chapter, it is shown that on a parallel architecture, waveform methods can be even faster.

A particularly attractive attribute of the waveform methods is that they are easily parallelized. To demonstrate this, the waveform algorithms described in the previous chapters were implemented in the pWORDS program. Because they are based on a block iterative method, where the iterates are waveforms over the entire simulation interval $[0, T]$, the waveform methods are best suited to a coarse-grained Multiple-Instruction-Multiple-Data (MIMD) type of architecture. Accordingly, the two target architectures used in the pWORDS program are the Intel iPSC/860 hypercube and a pair of workstations running the parallel virtual machine software (PVM).

Since the specifics of a parallel machine can (and should) influence the design of a program, we begin, in Section 6.2 by describing the characteristics of the two machines. Following this, in Section 6.3, the implementation details of the pWORDS program are given. In particular, this section contains an outline of the message-passing structure of the waveform and pointwise codes. Finally, in Section 6.4, comparative timing results for a suite of examples are presented.

6.2 Parallel Machines

This section is a brief description of the two message-passing, coarse-grained parallel machines used for the pWORDS experiments. From a programming standpoint the

two architectures are very similar, since they are both based on the MIMD model of computation, with message-passing communication and the C language. And for both architectures, a potentially large number of processors work concurrently on pieces of a single problem. Provided that communication between the compute nodes is kept to a minimum, this concurrency can lead to a substantial speedup.

6.2.1 Intel iPSC/860

The Intel iPSC/860 used in the experiments consisted of 32 compute nodes (Intel i860 processors each with 8 megabytes of memory) connected in a hypercubic network and an additional separate host computer (Intel386 processor), as an interface to the compute nodes. The host computer is used to coordinate the computations being performed by the compute nodes. In an iPSC, the compute nodes are independent processor/memory pairs, with distinct memory spaces. Communication between processors and to the host is based on message-passing. Although the hypercubic connection implies that communication is fastest between nearest neighbors, every compute node can send messages to any other compute node or to the host. Likewise, the host can send messages to any particular compute node, or broadcast them to all of the compute nodes. A message-passing library written in the C language was used in pWORDS.

6.2.2 PVM Cluster

The parallel virtual machine (PVM) software is a powerful tool that enables a cluster of workstations to function like a parallel architecture such as the Intel hypercube. Like the hypercube, a PVM cluster consists of a workstation designated as the host, serving as an interface to a group of other workstations serving as compute nodes. Communication between workstations takes place on the Ethernet, via message passing routines. From a programmer's point of view, this gives a PVM cluster the look and feel of a real parallel machine.

Once the PVM daemon is started on the host machine, a subordinate PVM process is started on each of the compute node workstations. This PVM process runs invisibly in the background, coordinating the communication between the host and nodes. At the instruction of the host machine, each node loads a node program, also in the background, and begins concurrent computation.

One advantage of PVM is that the cluster can consist of a wide variety of different types of workstations, each with its own memory and file system. The memory is particularly important for a waveform method, since the waveform iterates can consume many

kilobytes, or even megabytes. Another advantage of PVM is that it provides a tool for utilizing the idle cycles of any cluster of workstations. Finally, since the compute nodes are UNIX workstations with standard programming environments, standard tools can be used to debug the parallel code. Of course, a disadvantage of PVM is that communication between node and host can be significantly slower than that of a machine with specialized dedicated hardware.

For the pWORDS experiments, a pair of IBM RS/6000 Model 540 workstations were used as compute nodes, and an IBM RS/6000 Model 320H was used as a host machine. Each of the compute nodes had 256 megabytes of memory.

6.3 Parallel Implementation

Various parallel solution methods have been implemented in the WR-based device transient simulation program pWORDS, for computation on the Intel iPSC/860 and a cluster of workstations running the Parallel Virtual Machine (PVM) software. In this section the partitioning and communication patterns used for the parallel waveform methods are briefly described, and for comparison, an accelerated pointwise solution method using parallel Newton-GMRES iterations is also described. In the MIMD model of parallel computation offered by the iPSC/860 or the PVM cluster, a program is typically divided into two separate executables — a host program and a compute node program. The host program is responsible for assigning tasks to the compute nodes, and gathering their results when they are finished. The program running on each of the compute nodes performs the concurrent numerical computation.

6.3.1 Parallel Waveform Methods

As described in Section 4.2.1, to accelerate convergence, the WR methods in the WORDS program are based on dividing a device mesh into a linear sequence of vertical line blocks, that is, all of the nodes in each vertical mesh line are solved simultaneously. Furthermore, the blocks are processed in red/black order (see Sections 2.2.2–2.2.3). This leads to a block consistently ordered problem that can be solved with the convolution SOR method, described in Chapter 5.

The advantage of the red/black ordering scheme for relaxation on a parallel machine, is that during each iteration, no block depends on the solution of another block of the same color. In other words, the red blocks depend only on the solutions of the neighboring black blocks, and the black blocks depend only on the solutions of the neighboring red

blocks. This implies that once the black block solutions of the previous iteration have been communicated to the red blocks, all of the red blocks can be solved in parallel, with no other synchronization. Similarly, once the red blocks have all been solved, and their solutions have been communicated to the black blocks, all of the black blocks can be solved in parallel, with no other synchronization.

To begin a parallel WR computation, the host program reads in the device input file that specifies the device geometry and discretization mesh as well as the voltage boundary conditions imposed on the device. This information is broadcast to the compute nodes. The host program then partitions the device mesh, and assigns portions of the mesh to each compute node. A device mesh consisting of N vertical-line blocks is partitioned into $N/2$ pairs of adjacent red and black lines, and each of $N/2$ compute nodes is assigned a pair of lines. It is trivial to embed the linear sequence of red/black pairs of lines into the hypercubic network of the iPSC/860 so that all communication between compute nodes is between nearest neighbors [17]. This embedding ensures that compute node i will only communicate with its nearest neighbors, compute nodes $i-1$ and $i+1$.

At the start of the compute node program, the compute nodes receive the pair of vertical lines assigned by the host. In addition to this pair of lines, each compute node also contains storage for the two vertical lines on either side of the contiguous block. These "pseudo-lines" are used only to store the solutions generated and communicated by the compute nodes solving those adjacent vertical lines.

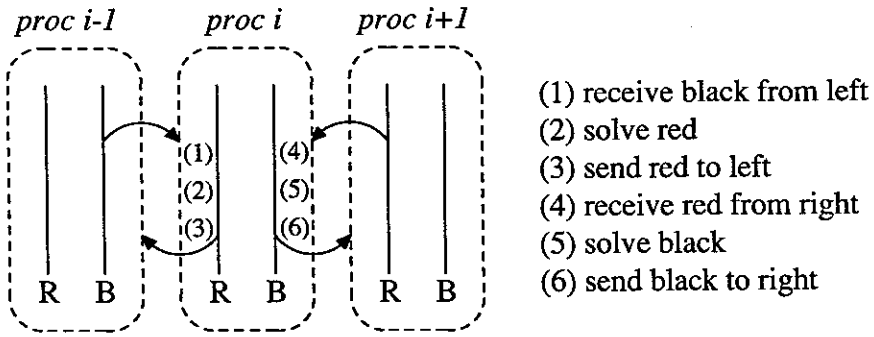

FIGURE 6-1: Illustration of the communication and computation steps performed by compute node i during one parallel waveform method iteration.

In each iteration of the parallel algorithm, the $N/2$ compute nodes first solve the $N/2$ red lines concurrently, and then solve the $N/2$ black lines concurrently. Figure 6-1 is an illustration of the communication and computation steps performed by compute node i to complete one parallel waveform method iteration. Note that the red line of compute node i has the black line of compute node $i-1$ as its left boundary condition. Accordingly, the first step of the red line solution process is for each compute node i

ALGORITHM 6.3.1 (OUTLINE OF PARALLEL RED/BLACK BLOCK WR).

Choose initial guess waveforms that satisfy initial conditions.

Send the black line solution to the right.

Iterate:

1. *Receive* black line solution from the left (BLOCKING).
2. *Solve* the device equations (1.9)–(1.11) for the red line.
3. *Send* the red line solution to the left.
4. *Receive* red line solution from the right (BLOCKING).
5. *Solve* the device equations (1.9)–(1.11) for the black line.
6. *Send* the black line solution to the right.

to receive the black line waveform solution sent by compute node $i-1$. (It is assumed that at the end of the previous iteration, each compute node sent its black line waveform solution to the right.) The black line that is the right boundary condition of the red line of compute node i is already resident within the node, so that the compute node can now solve its red line. Once its red line is solved, each compute node immediately sends its red line waveform solution leftward. This completes the first half of the iteration, the solution and communication of the red lines.

Solving the black lines in the second half of the iteration requires a similar communication pattern. Each compute node i receives the red line waveform solution sent from the right, and solves its black line. Then each compute node sends its black line waveform solution to the right, completing the solution and communication of the black blocks.

An outline of the compute node algorithm for a red/black block waveform method is shown in Algorithm 6.3.1. Each iteration of the parallel waveform algorithm contains only two blocking communications — before solving a line, each compute node must wait to receive the waveform of a neighboring line. The algorithm therefore requires very little synchronization between the compute nodes, and the communication that is required consists of large packets of information, entire waveforms.

If fewer than $N/2$ compute nodes are available, then each compute node is given multiple pairs of red/black lines that are adjacent in the device mesh. This implies that some of the lines (both red and black) residing on a compute node will depend only on other lines residing in the compute node, as shown in Figure 6-2. In this case, communication and computation can be overlapped — the red lines that do not depend on

other compute node solutions are solved before waiting to receive the black line solutions. Similarly, the black lines that do not depend on other compute node solutions are solved before waiting to receive the red line solutions.

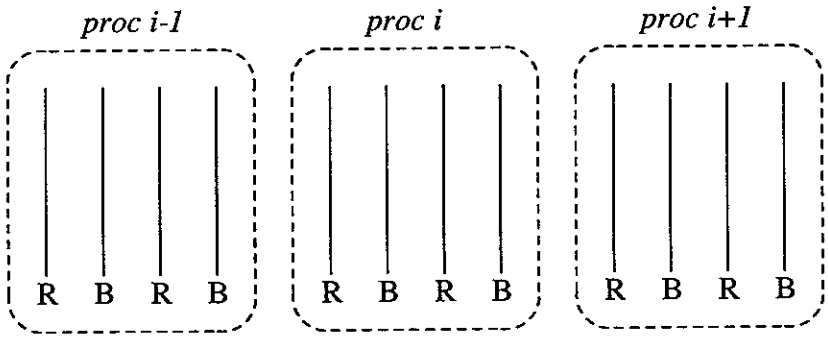

FIGURE 6-2: When less than $N/2$ compute nodes are available, each may be assigned several adjacent pairs of red/black lines.

6.3.2 Pointwise GMRES

A particularly effective serial algorithm, used to obtain the results of Section 5.7, is the pointwise Newton-GMRES algorithm in which a preconditioned GMRES algorithm [35] is used to solve the linear systems arising at each Newton iteration of each timestep of an implicit integration formula applied to the device system (1.9)–(1.11). To partition the problem for a parallel implementation, the pointwise Newton-GMRES method in pWORDS uses the same vertical-line blocks as in the waveform methods. The blocking is then used to partition the block tridiagonal matrix of the whole problem.

First, to simplify comparisons to the parallel waveform methods, the host assigns pairs of vertical-line blocks to the compute nodes exactly as in the waveform method. Given a particular block, the corresponding compute node is responsible for the storage and computation of the corresponding pieces of the matrix and GMRES vectors in that "block row", as shown in Figure 6-3. For the block tridiagonal matrix of the whole problem, this implies that each compute node must generate and store the appropriate block diagonal pieces of the matrix, as well as the off-diagonal blocks for the block rows. Once assigned a particular block, each compute node is responsible for generating the corresponding piece of the vector resulting from the matrix-vector product.

Unfortunately, partitioning the matrix and the vectors implies that the parallel pointwise Newton-GMRES algorithm requires many communication steps, each consisting of relatively small packets. Before every Newton iteration at every timepoint, a compute node must receive the two vectors of solutions of the neighboring lines from the left

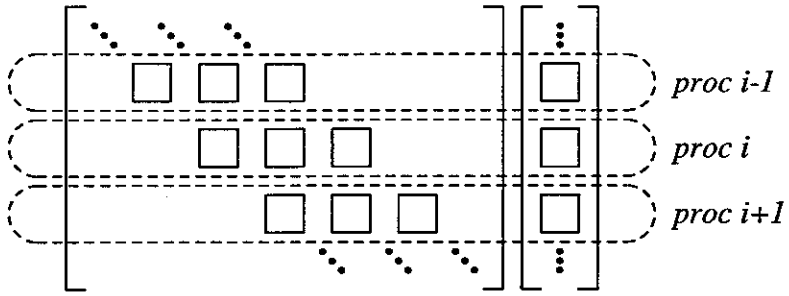

FIGURE 6-3: To partition the matrix-vector product, each processor is assigned the block rows corresponding to a pair of vertical line blocks.

and the right, in order to generate the block diagonal and block off-diagonal matrices. To accomplish the matrix-vector product for each GMRES iteration, a compute node must exchange pieces of the multiplicand vector with the neighboring compute nodes. Moreover, each GMRES iteration requires a number of inner product calculations, each of which requires a global sum. Although it may be possible to overlap these communication operations with local computation, a significant amount of inter-processor synchronization is still required. An approach such as that given in [46] might prove to be helpful, however.

6.4 Experimental Results

To compare the parallel performance of the convolution SOR accelerated waveform method and the pointwise Newton-GMRES algorithm, numerical experiments were conducted using the same eight examples used in Section 5.7. The three MOS devices of Table 5-1 were used to construct six simulation examples, each device being subjected to either a drain voltage pulse with the gate held high (the D examples), or a gate voltage pulse with the drain held high (the G examples). In addition, to observe the effect on WR convergence, two additional gate-pulsed examples were constructed by refining the device meshes of karG and soiG to contain 64 vertical lines. The drain-driven karD test setup is illustrated in Figure 5-7.

As in Section 5.7, to simplify comparisons between methods, the backward Euler method using 256 fixed timesteps was used for all experiments, on a simulation interval of 51.2 or 512 picoseconds. The convergence criterion for all experiments was the requirement that the maximum error over the simulation interval in the value of any terminal current be less than one part in 10^4. To provide an initial guess for WRN (WR with a single waveform Newton iteration) and for WRN with convolution SOR acceleration, 16

or 32 initial WR iterations were used.

Although using globally uniform timesteps eliminates the ability of waveform to exploit multirate behavior, it also avoids the issue of load balancing for the parallel waveform methods, since each vertical-line block will require nearly the same amount of work. The following results will show that even without the multirate advantage, accelerated waveform methods can be significantly faster than pointwise methods on a parallel processor.

Table 6-1 contains the results of the parallel experiments conducted on the pair of model 540 workstations using PVM. The table shows the time required for solution on a serial machine (one of the PVM workstations) compared to the time required for solution on the pair of workstations. To ensure a meaningful comparison, the times were measure in elapsed wall clock seconds.

		convolution SOR		Newton-GMRES		GMRES
example	(size)	serial	PVM (2)	serial	PVM (2)	iters
lddD	(656)	584.81	303.28	620.00	10909.46	53741
lddG	(656)	360.15	184.56	505.92	9017.11	43680
soiD	(856)	388.62	198.96	188.53	2387.35	12363
soiG	(856)	369.16	188.67	205.28	2609.86	13466
karD	(1379)	1088.41	553.84	673.81	5456.94	26022
karG	(1379)	712.74	365.41	794.01	6522.62	30858
soiG64	(2292)	2239.39	1117.39	2249.01	NA	48176
karG64	(2854)	3328.85	1685.44	5205.89	NA	71316

Table 6-1: Comparison of parallel and serial times for the convolution SOR and pointwise Newton-GMRES methods, on a PVM cluster of 2 workstations. The time required by the pointwise Newton-GMRES method on the PVM cluster was roughly proportional to the total number of GMRES iterations.

In Table 6-1, the waveform method shows an encouraging and remarkable scalability — in every instance, the simulation time required by the PVM cluster was about half of that required by a serial machine. This success can be attributed to the infrequent communication inherent in the parallel waveform algorithm, as well as to the overlap of communication and computation enabled by the multiple pairs of red/black lines residing on each compute node.

On the other hand, Table 6-1 shows that the parallel pointwise Newton-GMRES method actually became significantly slower on the PVM cluster. This can be attributed to the many small communications and synchronizations required at each time point in the simulation interval. Since the PVM cluster communication is based on the Ethernet, the frequent message-passing is obviously expensive. For the parallel Newton-GMRES method, the amount of communication is proportional to the number of GMRES itera-

tions, and as Table 6-1 indicates, the time required for parallel Newton-GMRES on the PVM cluster was roughly proportional to the total number of GMRES iterations. Clearly, the parallel Newton-GMRES method is bounded by the cost of its communications, at every iteration, at every timestep.

To further test the scalability of the waveform methods, Tables 6-2 and 6-3 show the CPU times required for solution of the eight examples on the Intel iPSC/860. The times shown are measured elapsed times on the compute nodes. The waveform method displays a remarkable scalability, with a roughly linear speedup up to 32 processors (the soiG64 and karG64 examples). As on the PVM cluster, the pointwise Newton-GMRES algorithm became slower on more than one processor, despite the dedicated communication hardware in the iPSC/860. Table 6-4 summarizes the best timing results for each method on the iPSC/860.

example	(mesh)	8 blk/proc	(procs)	4 blk/proc	(procs)	2 blk/proc	(procs)
lddD	(15×20)	NA		4721.25	(5)	2473.97	(10)
lddG	(15×20)	NA		2190.19	(5)	1182.51	(10)
soiD	(18×24)	1871.74	(3)	1037.27	(6)	554.67	(12)
soiG	(18×24)	1504.04	(3)	812.72	(6)	429.44	(12)
karD	(19×31)	4378.82	(4)	2256.21	(8)	1224.32	(16)
karG	(19×31)	3745.48	(4)	1939.39	(8)	1041.91	(16)

Table 6-2: Waveform Relaxation Newton timing results on the iPSC/860.

example	(mesh)	8 blk/proc	(procs)	4 blk/proc	(procs)	2 blk/proc	(procs)
lddD	(15×20)	NA		382.77	(5)	201.23	(10)
lddG	(15×20)	NA		235.67	(5)	128.59	(10)
soiD	(18×24)	405.93	(3)	225.41	(6)	120.15	(12)
soiG	(18×24)	417.40	(3)	225.74	(6)	117.31	(12)
karD	(19×31)	895.50	(4)	460.27	(8)	248.93	(16)
karG	(19×31)	590.88	(4)	308.29	(8)	165.61	(16)
soiG64	(18×64)	987.17	(8)	507.20	(16)	260.13	(32)
karG64	(19×64)	1386.32	(8)	713.97	(16)	373.53	(32)

Table 6-3: Convolution SOR timing results on the iPSC/860.

It is clear that the parallel implementations of both the waveform and the pointwise methods can be improved. The pointwise Newton-GMRES code can be rewritten to avoid some communication with the 386 host, perhaps by choosing one of the i860 compute nodes to act as coordinator. In addition, some of the Newton-GMRES communication can be overlapped with computation. The convolution SOR code can be improved by

example	(size)	Newton-GMRES	(procs)	conv SOR	(procs)
lddD	(656)	1184.68	(1)	201.23	(10)
lddG	(656)	989.74	(1)	128.59	(10)
soiD	(856)	366.11	(1)	120.15	(12)
soiG	(856)	401.42	(1)	117.31	(12)
karD	(1379)	1262.49	(1)	248.93	(16)
karG	(1379)	1492.17	(1)	165.61	(16)
soiG64	(2292)	4400 est.	(1)	260.13	(32)
karG64	(2854)	9800 est.	(1)	373.53	(32)

Table 6-4: Summary of the best timing results for each method on the iPSC/860.

using pointwise Newton-GMRES to solve each block, rather than sparse Gaussian elimination. But none of these improvements will change the inherent difference between the pointwise and the waveform algorithm: the pointwise algorithm requires much more synchronization and many more, smaller, communications.

7

Conclusions

This thesis investigated accelerated waveform relaxation techniques for the parallel transient simulation of semiconductor devices. After a review of standard numerical techniques in Chapter 2, the convergence of WR for the device problem was guaranteed in Chapter 3. A summary was given in Chapter 4 of some of the implementation issues encountered in the WR-based device simulation program WORDS, along with experimental results indicating that accelerated WR methods can be competitive with standard pointwise methods. The primary theoretical contribution of this thesis was the introduction and development in Chapter 5 of the convolution SOR technique for accelerating the convergence of waveform relaxation. Finally, in Chapter 6, a parallel implementation of the device WR algorithm was presented, along with results clearly demonstrating the superiority of waveform methods on parallel machines.

Although the convolution SOR theory does not directly apply to the device simulation problem, the CSOR technique was shown to dramatically accelerate the convergence rate of WR for devices. Because of its use of a convolution sum, the CSOR method correctly accounts for the frequency-dependence of the spectrum of the Gauss-Jacobi WR operator (e.g., Gauss-Jacobi WR smoothes high frequency components of the error waveform more rapidly than low frequency components), by in effect, using a different SOR parameter for each frequency. Apparently, the CSOR method inherits some of the robustness of the algebraic SOR method, and the optimal parameter formula (5.36) can be successfully applied to a wider class of problems.

The comparison of accelerated waveform relaxation algorithms to pointwise methods showed that accelerated waveform methods are competitive with standard pointwise methods on serial machines, and that accelerated waveform methods are significantly faster on commonly available loosely-coupled MIMD machines. In particular, parallel

accelerated waveform methods achieved a nearly linear speed-up on the 32 processor Intel iPSC/860 hypercube, whereas parallel versions of standard pointwise methods did not exhibit any parallel speed-up. The importance of the results is the strong implication that, as MIMD machines and cluster-based computing become more prevalent, accelerated waveform methods will gain in importance for all areas of simulation requiring the solution of initial value problems.

Future work is focused on refining the theory for CSOR, developing a multirate implementation of pWORDS, studying the behavior of accelerated waveform methods on other parallel machines, and applying accelerated waveform methods to new application areas.

Bibliography

[1] A. AGARWAL, *Limits on interconnection network performance*, IEEE Trans. Parallel Distrib. Sys., (1991), pp. 398–412.

[2] R. E. BANK, W. C. COUGHRAN, JR., W. FICHTNER, E. GROSSE, D. ROSE, AND R. SMITH, *Transient simulation of silicon devices and circuits*, IEEE Trans. CAD, 4 (1985), pp. 436–451.

[3] R. E. BANK AND D. J. ROSE, *Global approximate newton methods*, Numerische Mathematik, 37 (1981), pp. 279–295.

[4] R. K. BRAYTON, *Error estimates for the variable-step backward differentiation methods*, Tech. Report RC 6205 (#26655) Mathematics, IBM Thomas J. Watson Research Center, Yorktown Heights, New York, September 1976.

[5] R. K. BRAYTON, F. G. GUSTAVSON, AND G. D. HACHTEL, *A new efficient algorithm for solving differential-algebraic systems using implicit backward differentiation formulas*, Proc. IEEE, 60 (1972), pp. 98–108.

[6] R. K. BRAYTON AND C. H. TONG, *Stability of dynamical systems: A constructive approach*, IEEE Trans. Circuits Sys., CAS-26 (1979), pp. 224–234.

[7] K. E. BRENAN, S. L. CAMPBELL, , AND L. R. PETZOLD, *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*, Elsevier Science Publishing Co., New York, 1989.

[8] R. C. Y. CHIN AND T. A. MANTEUFFEL, *An analysis of block successive over-relaxation for a class of matrices with complex spectra*, SIAM J. Numer. Anal., 25 (1988), pp. 564–585.

[9] G. DAHLQUIST AND Å. BJÖRCK, *Numerical Methods*, Automatic Computation, Prentice-Hall, Englewood Cliffs, NJ, 1974.

[10] I. S. DUFF, A. M. ERISMAN, AND J. K. REID, *Direct Methods for Sparse Matrices*, Clarendon Press, Oxford, 1986.

[11] C. W. GEAR, *Numerical Initial Value Problems in Ordinary Differential Equations*, Automatic Computation, Prentice-Hall, Englewood Cliffs, NJ, 1971.

[12] G. H. GOLUB AND C. F. VAN LOAN, *Matrix Computations*, The John Hopkins University Press, Baltimore, Maryland, 1983.

[13] L. A. HAGEMAN AND D. M. YOUNG, *Applied Iterative Methods*, Academic Press, Orlando, FL, 1981.

[14] E. I. JURY, *Theory and Application of the z-Transform Method*, John Wiley & Sons, Inc., New York, 1964.

[15] L. V. KANTOROVICH AND G. P. AKILOV, *Functional Analysis in Normed Spaces*, Pergammon Press, Oxford, 1964.

[16] J. D. LAMBERT, *Computational Methods in Ordinary Differential Equations*, John Wiley & Sons, London, 1973.

[17] F. T. LEIGHTON, *Introduction to Parallel Algorithms and Architectures*, Morgan Kaufmann Publishers, San Mateo, CA, 1992.

[18] 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. CAD, 1 (1982), pp. 131–145.

[19] C. LUBICH AND A. OSTERMAN, *Multi-grid dynamic iteration for parabolic equations*, BIT, 27 (1987), pp. 216–234.

[20] A. LUMSDAINE, *Theoretical and Practical Aspects of Parallel Numerical Algorithms for Initial Value Problems, with Applications*, PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 1992.

[21] A. LUMSDAINE, M. REICHELT, AND J. WHITE, *Conjugate direction waveform methods for transient two-dimensional simulation of MOS devices*, in Proc. International Conference on Computer-Aided Design, Santa Clara, California, November 1991, pp. 116–119.

[22] A. LUMSDAINE AND M. W. REICHELT, *Waveform iterative techniques for device transient simulation on parallel machines*, in Proc. SIAM Conf. Parallel Proc. for Sci. Comp., March 1993.

[23] K. MAYARAM AND D. PEDERSON, *CODECS: A mixed-level device and circuit simulator*, in Proc. International Conference on Computer-Aided Design, Santa Clara, California, November 1988, pp. 112–115.

[24] U. MIEKKALA AND O. NEVANLINNA, *Convergence of dynamic iteration methods for initial value problems*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. 459–482.

[25] W. E. MILNE, *Numerical Solution of Differential Equations*, John Wiley & Sons, New York, 1953.

[26] R. S. MULLER AND T. I. KAMINS, *Device Electronics for Integrated Circuits*, John Wiley and Sons, New York, 1986.

[27] A. V. OPPENHEIM AND R. W. SCHAFER, *Discrete-Time Signal Processing*, Prentice Hall, Englewood Cliff, New Jersey, 1989.

[28] J. M. ORTEGA, *Numerical Analysis: A Second Course*, Academic Press, New York, 1972.

[29] J. M. ORTEGA AND W. C. RHEINBOLT, *Iterative Solution of Nonlinear Equations in Several Variables*, Computer Science and Applied Mathematics, Academic Press, New York, 1970.

[30] L. REICHEL AND L. N. TREFETHEN, *Eigenvalues and pseudo-eigenvalues of toeplitz matrices*, Linear Algebra and its Applications, 162–164 (1992), pp. 153–185.

[31] M. REICHELT, *Optimal convolution SOR acceleration of waveform relaxation with application to semiconductor device simulation*, in Proc. Copper Mountain Conference on Multigrid Methods, 1993.

[32] M. REICHELT, F. ODEH, AND J. WHITE, *A-stability of multirate integration methods, with application to parallel semiconductor device simulation*, in Proc. SIAM Conf. Parallel Proc. for Sci. Comp., March 1993.

[33] M. REICHELT, J. WHITE, AND J. ALLEN, *Waveform relaxation for transient two-dimensional simulation of MOS devices*, in Proc. International Conference on Computer-Aided Design, Santa Clara, CA, November 1989, pp. 412–415.

[34] M. REICHELT, J. WHITE, J. ALLEN, AND F. ODEH, *Waveform relaxation applied to transient device simulation*, in Proc. International Symposium on Circuits and Systems, Espoo, Finland, 1988.

[35] Y. SAAD AND M. SCHULTZ, *GMRES: A generalized minimum residual algorithm for solving nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 856–869.

[36] E. B. SAFF AND A. D. SNIDER, *Fundamentals of Complex Analysis for Mathematics, Science, and Engineering*, Prentice-Hall, Inc., New Jersey, 1976.

[37] R. SALEH AND J. WHITE, *Accelerating relaxation algorithms for circuit simulation using waveform-newton and step-size refinement*, IEEE Trans. CAD, 9 (1990), pp. 951–958.

[38] R. A. SALEH AND A. R. NEWTON, *Mixed-Mode Simulation*, The Kluwer International Series in Engineering and Computer Science, Kluwer Academic Publishers, Boston, 1990.

[39] A. SANGIOVANNI-VINCENTELLI, *Circuit simulation*, in Computer Design Aids for VLSI Circuits, P. Antognetti, D. O. Pederson, and H. de Man, eds., Martinus Nijhoff Publishers, The Hague, 1984, pp. 19–112.

[40] D. L. SCHARFETTER AND H. K. GUMMEL, *Large-signal analysis of a silicon read diode oscillator*, IEEE Trans. Electron Devices, ED-16 (1969), pp. 64–77.

[41] S. SELBERHERR, *Analysis and Simulation of Semiconductor Devices*, Springer-Verlag, New York, 1984.

[42] R. D. SKEEL, *Waveform iteration and the shifted Picard splitting*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 756–776.

[43] D. SMART AND J. WHITE, *Reducing the parallel solution time of sparse circuit matrices using reordered gaussian elimination and relaxation*, in Proc. International Symposium on Circuits and Systems, Espoo, Finland, June 1988, pp. 627–630.

[44] J. STOER AND R. BULIRSCH, *Introduction to Numerical Analysis*, Springer-Verlag, New York, 1980.

[45] G. STRANG, *Linear Algebra and Its Applications*, Academic Press, New York, 1980.

[46] E. D. STURLER, *A parallel restructured version of GMRES(m)*, in Proceedings of the Copper Mountain Conference on Iterative Methods, Copper Mountain, Colorado, 1992.

[47] L. N. TREFETHEN, *Pseudospectra of matrices*, in Numerical Analysis 1991, D. F. Griffiths and G. A. Watson, eds., Longman Scientific & Technical, Harlow, Essex, England, 1992.

[48] R. S. VARGA, *Matrix Iterative Analysis*, Automatic Computation Series, Prentice-Hall, Englewood Cliffs, NJ, 1962.

[49] M. VIDYASAGAR, *Nonlinear Systems Analysis*, Prentice-Hall, Englewood Cliffs, NJ, 1978.

[50] J. K. WHITE AND A. SANGIOVANNI-VINCENTELLI, *Relaxation Techniques for the Simulation of VLSI Circuits*, The Kluwer International Series in Engineering and Computer Science, Kluwer Academic Publishers, Boston, 1987.

[51] D. M. YOUNG, *Iterative methods for solving partial difference equations of elliptic type*, Trans. Amer. Math. Soc., 76 (1954), pp. 92–111.

[52] ——, *Iterative Solution of Large Linear Systems*, Academic Press, Orlando, FL, 1971.