Theoretical and Practical Aspects of Parallel Numerical Algorithms for Initial Value Problems, with Applications

RLE Technical Report No. 574

Andrew Lumsdaine

September 1992

Research Laboratory of Electronics
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139-4307

Theoretical and Practical Aspects of Parallel Numerical Algorithms for Initial Value Problems, with Applications

RLE Technical Report No. 574

Andrew Lumsdaine

September 1992

Research Laboratory of Electronics
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139-4307

This work was supported in part by IBM Corporation, in part by the Defense Advanced Research Projects Agency under the Office of Naval Research grant N00014-91-J-1698 and under the National Science Foundation grant MIP-88-14612, and in part by an AEA/Dynatech faculty development fellowship.

Theoretical and Practical Aspects of Parallel Numerical Algorithms for Initial Value Problems, with Applications

by

Andrew Lumsdaine

Submitted to the Department of Electrical Engineering and Computer Science
on January 2, 1992, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

Theoretical and practical aspects of parallel numerical methods for solving initial value problems are investigated, with particular attention to two applications from electrical engineering.

First, algorithms for massively parallel circuit-level simulation of the grid-based analog signal processing arrays currently being developed for robotic vision applications are described, and simulation results presented. The trapezoidal rule is used to discretize the differential equations that describe the analog array behavior, Newton's method is used to solve the nonlinear equations generated at each time-step, and a block preconditioned conjugate-gradient squared algorithm is used to solve the linear equations generated by Newton's method. Excellent parallel performance of the algorithm is achieved through the use of a novel, but very natural, mapping of the circuit data onto a massively parallel architecture. The mapping takes advantage of the underlying computer architecture and the structure of the analog array problem. Experimental results demonstrate that a full-size Connection Machine can provide a 650 times speedup over a SUN-4/490 workstation.

Next, a new conjugate direction algorithm for accelerating waveform relaxation applied to the semiconductor device transient simulation problem is developed. A Galerkin method is applied to solving the system of second-kind Volterra integral equations which characterize the classical dynamic iteration methods for the linear time-varying initial value problem. It is shown that the Galerkin approximations can be computed iteratively using conjugate-direction algorithms. The resulting iterative methods are combined with an operator Newton method and applied to solving the nonlinear differential-algebraic system generated by spatial discretization of the time-dependent semiconductor device equations. Experimental results are included which demonstrate the conjugate-direction methods are significantly faster than classical dynamic iteration methods.

The results from both applications are encouraging and demonstrate that for specific initial value problems, the largest performance gains can be achieved by using closely matched algorithms and architectures to exploit characteristic features of the particular problem to be solved.

Thesis Supervisor: Jacob K. White
Title: Associate Professor

Thesis Supervisor: John L. Wyatt, Jr.
Title: Professor

To Wendy, with all my love

Acknowledgments

It was my profound privilege to have *two* supervisors, Prof. Jacob White and Prof. John Wyatt, directing the research contained in this thesis. Not only did they direct my work to ensure its quality, but, more importantly, they taught me how to direct myself. I must add that having two supervisors, especially two of the caliber of Profs. White and Wyatt, has benefited me far more than twice as much as having a single supervisor.

I am grateful to several fellow graduate students without whose help I never would have finished the work in this thesis. Foremost I must acknowledge the extensive assistance I received from Miguel Silveira. Throughout our three-year relationship, we collaborated on several large projects, in particular SIMLAB and CMVSIM. These programs would never have been completed without him and would certainly not be as elegant without the refinement they obtained as a result of our constant wrangling over details. I worked very closely with Abe Elfadel on several vision related projects. Abe was always a knowledgeable resource for any sort of questions I had of a theoretical nature — and there were a lot of them. Mark Reichelt gave a large measure of his time in providing me with access to his *WORDS* program and in helping me with the implementation and experiments for the waveform conjugate direction methods. Bob Armstrong was always available when I needed a software or hardware guru — he developed and provided the lab with a variety of useful tools, many of which were used in the preparation of this document.

I would also like to acknowledge Thinking Machines Corporation and Rolf Fiebrich for the generous contribution of their resources, Prof. Zhengfang Zhou for several helpful discussions, the MIT Vision Chip group for providing the motivation for the work in Chapter 3, and all the other students and faculty in the VLSI CAD group for their support and friendship.

Finally, completion of this thesis would not have been possible without the love and encouragement I received from my family, especially from my wife Wendy. I thank her for being so patient during the course of my graduate program and for supporting me financially, emotionally, and spiritually during that time. Thanks also to my parents, Edward and Monika Lumsdaine, who, on top of all the support they have given me over the years, found the time to proofread this thesis.

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, and an AEA/Dynatech faculty development fellowship.

Contents

List of Figures

List of Tables

List of Acronyms

CD	Conjugate Direction
CG	Conjugate Gradient
CGNR	CG Applied to the Normal Equations
CGS	Conjugate Gradient Squared
CM	Connection Machine
DAE	Differential Algebraic Equation
GCR	Generalized Conjugate Residual
GMRES	Generalized Minimum Residual
GJ	Gauss-Jacobi
GS	Gauss-Seidel
ILUCG(S)	Incomplete LU Factorization Preconditioned CG(S)
IVP	Initial Value Problem
KCL	Kirchoff's Current Law
KVL	Kirchoff's Voltage Law
MICCG	Modified Incomplete Cholesky Factorization Preconditioned CG
MIMD	Multiple Instruction, Multiple Data
MOS	Metal-Oxide-Semiconductor
MOSFET	MOS Field Effect Transistor
NLR	Nonlinear Relaxation
ODE	Ordinary Differential Equation
PDE	Partial Differential Equation
QMR	Quasi-Minimal Residual
SIMD	Single Instruction, Multiple Data
VLSI	Very Large Scale Integration
WCGS	Waveform Conjugate Gradient Squared
WGCR	Waveform Generalized Conjugate Residual
WGMRES	Waveform Generalized Minimum Residual
WN	Waveform Newton
WR	Waveform Relaxation
WRN	Waveform Relaxation Newton

List of Symbols

Vectors

Members of a vector space are denoted with the bold math font, e.g., $\boldsymbol{x}$. The vector space in question may be $\mathbb{R}^n$, or a function space. Components of a multi-dimensional vector are denoted in the plain math font, with the appropriate subscript, e.g., x_i. If the vector space is additionally an inner product space, the inner product of $\boldsymbol{x}$ and $\boldsymbol{y}$ is denoted by $\langle \boldsymbol{x}, \boldsymbol{y} \rangle$. Constant vectors are sometimes indicated with a subscript, e.g., $\boldsymbol{x}_0$. The symbols for some particular vectors used in this thesis are:

$\boldsymbol{b}$ Right-hand side (as in $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$).

$\boldsymbol{e}$ Error. The elementary basis vectors for $\mathbb{R}^n$ are denoted $\boldsymbol{e}_1, \ldots, \boldsymbol{e}_n$.

$\boldsymbol{p}$ Search direction.

$\boldsymbol{r}$ Residual.

Matrices

Matrices are denoted in upper-case with the bold math font, e.g., $\boldsymbol{A}$. Components of a matrix are denoted in upper- or lower-case with the plain math font, using the appropriate subscript, e.g., A_{ij} or a_{ij}. The algebraic or Hermitian transpose of a matrix is denoted with a superscript $\dagger$, e.g., $\boldsymbol{A}^\dagger$. The identity matrix is denoted by $\boldsymbol{I}$.

Operators

Operators are denoted in upper-case with the script font, e.g., $\mathcal{A}$.

Spaces

Spaces are denoted with the blackboard bold math font, e.g., $\mathbb{R}^n$. The symbols for some particular spaces used in this thesis are:

$\mathbb{L}_2([0,T], \mathbb{R}^n)$ Function space of square-integrable functions (in the Lebesgue sense) mapping from the interval $[0,T]$ to $\mathbb{R}^n$.

$\mathbb{N}$ Set of natural numbers, i.e., $\{1, 2, \ldots\}$.

$\mathbb{R}^n$ n-dimensional Euclidean space.

Miscellaneous

The following is a brief description of other miscellaneous nomenclature:

^{k} The superscript k denotes the k^{th} iterate of the associated quantity, e.g., $\boldsymbol{x}^k$ is the k^{th} iterate of $\boldsymbol{x}$. The superscript k is the usually used for iterates with a linear iterative procedure, whereas the superscript m is usually used for iterates with a nonlinear iterative procedure. Indexing generally begins from $k = 0$. Occasionally, the superscript is also used for exponentiation, but those cases will be clear from the context.

$\operatorname{cl} A$ Closure of the set A.

ess.sup Essential supremum.

$\nabla\phi$ Gradient of the function ϕ.

$\Omega(\cdot)$ Lower bound complexity.

$\mathcal{O}(\cdot)$ Upper bound complexity.

1

Introduction

1.1 Initial Value Problems

Many interesting applications can be modeled as an initial value problem (IVP), e.g.,

$$\begin{aligned} \boldsymbol{F}(\boldsymbol{x}'(t), \boldsymbol{x}(t), t) &= 0 \\ \boldsymbol{x}(0) &= \boldsymbol{x}_0 \end{aligned} \tag{1.1}$$

where $\boldsymbol{x}(t) \in \mathbb{R}^n$ and $\boldsymbol{F} : \mathbb{R}^{2n+1} \to \mathbb{R}^n$. Here, (1.1) can describe an IVP for an ordinary differential equation (ODE) system or for a differential algebraic equation (DAE) system. It is assumed that The function $\boldsymbol{F}$ is such that the solution $\boldsymbol{x}(t)$ exists and is unique on a simulation interval of interest, say, $t \in [0, T]$, and that the initial condition is consistent for the DAE case.

In general, an analytic solution to (1.1) cannot be found, so the problem must be solved numerically. With the typical approach, (1.1) is first discretized in time with an integration method. Since DAE and stiff ODE systems require the use of implicit integration schemes, the time discretization will generate a sequence of nonlinear algebraic problems which are solved with an iterative method, usually a modified Newton method. The sequence of linear algebraic systems generated at each iteration of the nonlinear solution method are then solved with Gaussian elimination. The above process is the "implicit-integration, Newton, direct method" canon and forms the basis for most general-purpose codes for solving large-scale IVP's [1].

The standard approach has two computational bottlenecks (which bottleneck will dominate depends on the particular problem):

- Function evaluation — computation of $\boldsymbol{F}(\cdot)$ and the associated Jacobian $\boldsymbol{J_F}(\cdot)$. The cost of evaluating $\boldsymbol{J_F}$ grows with problem size and with the degree of coupling.

For densely coupled problems, evaluating $\boldsymbol{J}_F$ costs $\mathcal{O}(n^2)$ operations; for sparsely coupled problems, the cost of evaluating $\boldsymbol{J}_F$ can be as low as $\mathcal{O}(n)$.

- Linear system solution — solving the linear system at each iteration of the nonlinear solution process. The complexity of direct elimination methods for solving systems of equations is polynomial in n, typically from $\mathcal{O}(n^{1.5})$ for sparse problems to $\mathcal{O}(n^3)$ for dense problems.

When using the standard Newton method, the function evaluation and the linear system solution are performed at each iteration of each nonlinear system solution at each timestep. However, certain modified Newton methods recalculate the Jacobian only at certain intervals (e.g., every third Newton iteration) [2], thereby reducing the work required, although not the complexity, by a constant factor.

Efforts to improve the computational efficiency for numerically solving IVP's focus on improving the efficiency of the function evaluation and the linear system solution. These efforts fall into two general (overlapping) categories: algorithmic improvement and hardware improvement. For example, one could use an iterative method for the linear system solution, or implement the solver on a vector or parallel processing machine. For some problems, such an approach might be highly effective, but for other problems, such an approach might be a disaster. It seems that *general-purpose* codes for solving IVP's must follow the "implicit-integration, Newton, direct method" canon because such codes are written to be able to reliably handle the largest possible class of problems. However, this formula can be quite limiting for *specific* problems that can benefit from the application of more specialized algorithms.

Therefore, the following observation is made, which is the theme of this thesis: *For specific initial value problems, one can achieve the largest performance gains by using closely matched algorithms and architectures to exploit characteristic features of the particular problem to be solved.*

This thesis will examine methods for solving two initial value problems from electrical engineering — a circuit simulation problem and a device simulation problem. The attacks on these problems will be two-fold. First, methods suitable for implementation on parallel machines will be developed. Second, it will be attempted to exploit the problems fully through the use of sophisticated numerical techniques.

1.2 A Circuit Simulation Problem

The nodal analysis formulation of the circuit transient simulation problem is described by

$$\begin{aligned} \tfrac{d}{dt}\boldsymbol{q}\left(\boldsymbol{v}(t),t\right)+\boldsymbol{i}\left(\boldsymbol{v}(t),t\right) &= 0 \\ \boldsymbol{v}(0) &= \boldsymbol{v}_0 \end{aligned} \tag{1.2}$$

where $\boldsymbol{v}(t), \boldsymbol{q}\left(\boldsymbol{v}(t),t\right), \boldsymbol{i}\left(\boldsymbol{v}(t),t\right) \in \mathbb{R}^n$ are the vectors of node voltages, sums of node charges, and sums of node resistive currents, respectively, and n is the total number of nodes in the circuit.

Numerical techniques for solving (1.2) are very well developed — for all practical purposes, the *general* circuit simulation problem has been solved [3, 4]. Programs like SPICE [5] or ASTAP [6] — which follow the "implicit-integration, Newton, direct method" approach to solving (1.2) — are capable of simulating virtually any circuit, given enough time. Unfortunately, for some types of circuits, "enough time" is too much time for simulation to be a practical part of a circuit design cycle.

Robotic vision circuits [7, 8] form one such class of circuits which are intractable for standard analog circuit simulators such as SPICE or ASTAP. The vision circuits are necessarily very large and must be simulated at the analog level (i.e., one cannot perform simulations at a switch or gate level as is commonly done with very large digital circuits). Standard analog circuit simulators are not able to handle vision circuits simply because of their immense size, since the computation time for these simulators grows super-linearly with the size of the circuit. This super-linear growth is particularly pronounced for vision circuits because their structure produces a Jacobian matrix that generates much more fill during Gaussian elimination than do generic circuits having the same number of nodes.

Although the structure of the vision chips is disadvantageous for a direct-methods solver, it is advantageous for other algorithms and for certain parallel architectures. In particular, the regular structure of the problem implies that the simulation computations can be accelerated by a massively parallel SIMD computer, such as the Connection Machine®[9]. Moreover, the coupling between cells in the analog array is such that a block-iterative scheme can be used to solve the equations generated by an implicit time-discretization scheme.

Connection Machine is a registered trademark of Thinking Machines Corporation.

1.3 A Device Simulation Problem

The second problem to be examined is the semiconductor device transient simulation problem. For this problem, a new class of waveform methods, waveform conjugate direction methods, will be developed and applied. This approach is somewhat more general than that used for the circuit simulation problem, but there are specific features of this problem which make application of the new method effective. Moreover, details of implementation exploit other aspects of this problem in order to increase efficiency.

After standard spatial discretization on an n-node rectangular mesh, the semiconductor device transient simulation problem is modeled by a differential-algebraic system of $3n$ equations in $3n$ unknowns denoted by

$$\begin{aligned}
\boldsymbol{f}_1(\boldsymbol{u}(t),\boldsymbol{n}(t),\boldsymbol{p}(t)) &= 0\\
\boldsymbol{f}_2(\boldsymbol{u}(t),\boldsymbol{n}(t),\boldsymbol{p}(t)) &= \tfrac{d}{dt}\boldsymbol{n}(t)\\
\boldsymbol{f}_3(\boldsymbol{u}(t),\boldsymbol{n}(t),\boldsymbol{p}(t)) &= \tfrac{d}{dt}\boldsymbol{p}(t)\\
\boldsymbol{n}(0) &= \boldsymbol{n}_0\\
\boldsymbol{p}(0) &= \boldsymbol{p}_0\\
\boldsymbol{f}_1(\boldsymbol{u}(0),\boldsymbol{n}(0),\boldsymbol{p}(0)) &= 0
\end{aligned}$$

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, respectively, and $\boldsymbol{f}_1,\boldsymbol{f}_2,\boldsymbol{f}_3 : \mathbb{R}^{3n} \rightarrow \mathbb{R}^n$ are the Poisson, electron current continuity and hole current continuity equations, respectively.

The device transient simulation problem is studied in much detail in [10]. The approach used is to discretize the problem in time with a low-order implicit method, apply Newton's method, and use a direct-methods solver for the linear system solution. The use of iterative methods for the linear system solution is discussed in [11], but the reported results are discouraging. For two-dimensional simulations, the authors claim that direct methods are superior to iterative methods, but that iterative methods may be more effective with three-dimensional simulations.

A waveform relaxation (WR) based approach to the device transient simulation problem was introduced in [12]. This approach was shown to be computationally efficient compared to the traditional "implicit-integration, Newton, direct method" approach. However, the WR algorithm typically requires hundreds of iterations to achieve an accurate solution, which suggests that further performance gains can be realized by the application of methods for accelerating the convergence of the WR algorithm.

In this thesis, waveform conjugate-direction methods are developed for accelerating waveform relaxation applied to solving the linear, but possibly time-varying, initial value problem. The development of the waveform conjugate-direction methods proceeds in a few key steps. First, the linear initial value problem is converted to a system of second-kind Volterra integral equations through the use of a dynamic preconditioner. It is then shown that a Galerkin method can be used to solve this integral equation system and that certain conjugate direction methods can be used to generate the Galerkin approximations. The resulting method is combined with the waveform Newton method to produce a hybrid algorithm for solving nonlinear initial value problems. The hybrid method is then applied to solving the differential-algebraic system of equations that describe the device transient simulation problem.

1.4 Overview

A review of some of the most popular techniques for solving initial value problems is given in Chapter 2. In Chapter 3, algorithms are developed for CMVSIM, a program for simulating grid-based analog signal processor chips on the Connection Machine. The waveform conjugate direction methods are studied in Chapter 4, where the waveform generalized conjugate residual algorithm is developed and analyzed as a particular waveform conjugate direction method. The device transient simulation problem is described, and simulation results comparing conjugate direction algorithms with standard waveform techniques are presented.

References

[1] K. E. Brenan, S. L. Campbell, and L. R. Petzold, *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations.* New York: North Holland, 1989.

[2] J. M. Ortega and W. C. Rheinbolt, *Iterative Solution of Nonlinear Equations in Several Variables.* Computer Science and Applied Mathematics, New York: Academic Press, 1970.

[3] L. O. Chua and P.-M. Lin, *Computer-Aided Analysis of Electronic Circuits.* Englewood Cliffs, New Jersey: Prentice Hall, 1975.

[4] J. Vlach and K. Singhal, *Computer Methods for Circuit Analysis and Design.* Berkshire, England: Van Nostrand Reinhold, 1983.

[5] L. W. Nagel, "SPICE2: A computer program to simulate semiconductor circuits," Tech. Rep. ERL M520, Electronics Research Laboratory Report, University of California, Berkeley, Berkeley, California, May 1975.

[6] W. T. Weeks, A. J. Jimenez, G. W. Mahoney, D. Mehta, H. Quasemzadeh, and T. R. Scott, "Algorithms for ASTAP - A network analysis program," *IEEE Transactions on Circuit Theory*, pp. 628–634, November 1973.

[7] C. Mead, *Analog VLSI and Neural Systems.* Reading, MA: Addison-Wesley, 1988.

[8] J. L. Wyatt Jr., *et al*, "Smart vision sensors: Analog VLSI systems for integrated image acquisition and early vision processing." Massachusetts Institute of Technology. Unpublished, 1988.

[9] W. D. Hillis, *The Connection Machine.* New Haven, CT: MIT Press, 1985.

[10] R. Bank, W. Coughran, Jr., W. Fichtner, E. Grosse, D. Rose, and R. Smith, "Transient simulation of silicon devices and circuits," *IEEE Trans. CAD*, vol. 4, pp. 436–451, October 1985.

[11] C. Rafferty, M. Pinto, and R. Dutton, "Iterative methods in semiconductor device simulation," *IEEE Trans. CAD*, vol. 4, pp. 462–471, October 1985.

[12] M. Reichelt, J. White, and J. Allen, "Waveform relaxation for transient two-dimensional simulation of MOS devices," in *International Conference on Computer Aided-Design*, (Santa Clara, California), pp. 412–415, November 1989.

2

Review of Numerical Techniques for Initial Value Problems

2.1 Introduction

There exist a myriad of approaches for solving initial value problems — a non-exhaustive taxonomy is presented in Figure 2-1. To solve the IVP, the system is first decomposed in time (for point-wise solution methods) or space (for waveform methods). The point-wise solution is computed with the application of an integration method, possibly followed by a nonlinear algebraic and linear algebraic solution step. The waveform methods treat the nonlinear IVP as a nonlinear problem on a function space — note the similarity of the taxonomies above and below "Discretize".

In this chapter, techniques for solving initial value problems are reviewed. Although the techniques are applied in top-down fashion (e.g., integration, nonlinear solution, linear solution), the techniques are presented in somewhat of a bottom-up order since the "top" algorithms generally build upon the "bottom" algorithms. The reader should refer to Figure 2-1 as a roadmap to keep the different algorithms in their proper context within the framework of solving initial value problems.

2.2 Linear Solution Methods

In this section, methods are reviewed for solving the n-dimensional linear system of equations

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

where $\boldsymbol{x}, \boldsymbol{b} \in \mathbb{R}^n$ and $\boldsymbol{A} : \mathbb{R}^n \rightarrow \mathbb{R}^n$ and is assumed to be non-singular.

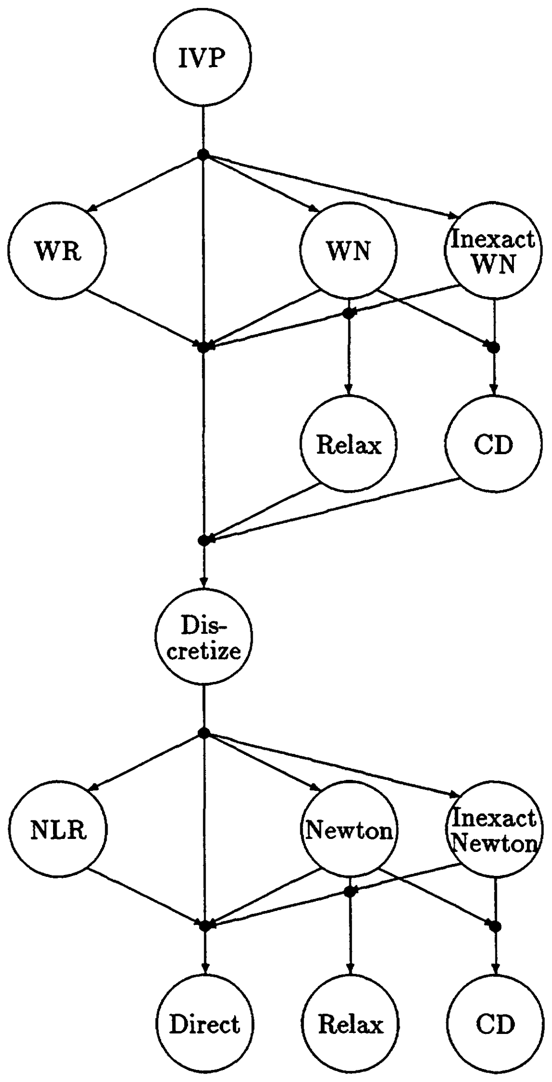

Figure 2-1: A (non-exhaustive) taxonomy of methods for solving IVP's. To solve the IVP, the system is first decomposed in time (for point-wise solution methods) or space (for waveform methods). The point-wise solution is computed with the application of an integration method, possibly followed by a nonlinear algebraic and linear algebraic solution step. The waveform methods treat the nonlinear IVP as a nonlinear problem on a function space — note the similarity of the taxonomies above and below "Discretize".

2.2.1 Direct Methods

The classical direct method for solving linear systems is Gaussian elimination, typically implemented with LU-factorization techniques. This algorithm decomposes the matrix $\boldsymbol{A}$ into lower and upper triangular factors $\boldsymbol{L}$ and $\boldsymbol{U}$, respectively, such that $\boldsymbol{A} = \boldsymbol{LU}$. The solution $\boldsymbol{x}$ to (2.1) is computed by first solving $\boldsymbol{Ly} = \boldsymbol{b}$ with a forward elimination process and then solving $\boldsymbol{Ux} = \boldsymbol{y}$ with backward substitution. A discussion of direct methods can be found in most linear algebra or numerical methods texts — see [1, 2] for dense matrix problems and [3] for sparse matrix problems.

The chief advantage of direct methods is reliability. With exact arithmetic, the solution $\boldsymbol{x}$ can be computed exactly in a fixed number of steps. However, direct methods have two major disadvantages: computational complexity and storage. The complexity of direct elimination methods for solving linear systems of equations is polynomial in N, typically from $\mathcal{O}(N^{1.5})$ for sparse problems to $\mathcal{O}(N^3)$ for dense problems[1]. For direct methods, the matrix itself must be stored in memory. This might not be particularly disadvantageous for the matrix itself, if the matrix is sparse. However, direct methods also require storage for the fill-in elements, i.e., matrix zero locations which become non-zero as the elimination process proceeds. Most iterative methods for solving linear systems only require that the matrix itself be stored, so the fill-in storage, which can be quite substantial, is not needed. Moreover, certain nonlinear solution methods, e.g., the so-called "matrix-free methods" do not even require an explicit representation of the matrix at all.

Relaxation and conjugate direction iterative methods for linear systems are presented in Sections 2.2.2 and 2.2.3. Matrix-free methods are discussed in Section 2.3.4.

2.2.2 Relaxation Methods

Linear relaxation methods seek to solve (2.1) by first decomposing the problem in space (i.e., pointwise) and then solving the decomposed problem in an iterative loop. The simplest relaxation method is the Richardson iteration [5, 6] which solves (2.1) by solving the following equations

$$x_i^{k+1} = x_i^k + b_i - \sum_{i=1}^{n} a_{ij} x_j^k$$

[1]$\mathcal{O}(N^3)$ is the commonly cited complexity for dense matrix problems. However, it is known that Gaussian elimination is not an optimal direct method. For instance, in [4], Strassen describes an algorithm for dense matrix problems having $\mathcal{O}(N^{2.8})$ complexity.

for each x_i^{k+1}, i.e., component i of $\boldsymbol{x}$ at iteration k. Two other popular relaxation methods are the Gauss-Jacobi and Gauss-Seidel algorithms which solve (2.1) by solving the equations

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

and

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

respectively, for x_i^{k+1} [6]. The above iterations can be described compactly in matrix form. Let $\boldsymbol{L}$, $\boldsymbol{U}$, and $\boldsymbol{D}$ be the strictly lower triangular, strictly upper triangular, and diagonal of $\boldsymbol{A}$, respectively. Then, the Richardson iteration can be expressed compactly as

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^k,$$

and the Gauss-Jacobi and Gauss-Seidel algorithms as

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

and

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

respectively.

It is interesting to note that the Gauss-Jacobi and Gauss-Seidel algorithms are essentially the Richardson iteration applied to a preconditioned form of (2.1). Consider the system of equations:

$$\boldsymbol{D}^{-1}\boldsymbol{A}\boldsymbol{x} = \boldsymbol{D}^{-1}\boldsymbol{b}$$

to which the Richardson iteration is applied:

$$\begin{aligned} \boldsymbol{x}^{k+1} &= \boldsymbol{x}^k + \boldsymbol{D}^{-1}\boldsymbol{b} - \boldsymbol{D}^{-1}\boldsymbol{A}\boldsymbol{x}^k \\ &= \boldsymbol{D}^{-1}\boldsymbol{b} - \boldsymbol{D}^{-1}(\boldsymbol{L} + \boldsymbol{U})\boldsymbol{x}^k \end{aligned}$$

which is precisely Gauss-Jacobi relaxation. Preconditioning with $(\boldsymbol{L} + \boldsymbol{D})^{-1}$ similarly yields the Gauss-Seidel algorithm.

In general, splittings of $\boldsymbol{A}$ can be described by letting $\boldsymbol{A} = \boldsymbol{M} - \boldsymbol{N}$ so that the generic relaxation method can be written as:

$$\boldsymbol{M}\boldsymbol{x}^{k+1} = \boldsymbol{N}\boldsymbol{x}^k + \boldsymbol{b}$$

or that

$$\boldsymbol{x}^{k+1} = \boldsymbol{M}^{-1}\boldsymbol{N}\boldsymbol{x}^k + \boldsymbol{M}^{-1}\boldsymbol{b}$$

Let $\boldsymbol{x}^*$ be the exact solution to (2.1) and define the error at the k^{th} iteration by $\boldsymbol{e}^k = \boldsymbol{x}^k - \boldsymbol{x}^*$. The error equation for the relaxation method is given by:

$$\boldsymbol{e}^{k+1} = \boldsymbol{M}^{-1}\boldsymbol{N}\boldsymbol{e}^k = \left(\boldsymbol{M}^{-1}\boldsymbol{N}\right)^{k+1}\boldsymbol{e}^0$$

The asymptotic convergence rate of linear relaxation is determined by the spectral radius of $\boldsymbol{M}^{-1}\boldsymbol{N}$. In order to guarantee convergence of the method for arbitrary $\boldsymbol{e}^0$, the spectral radius of $\boldsymbol{M}^{-1}\boldsymbol{N}$ must be strictly less than unity (see [6]).

2.2.3 Conjugate Direction Methods

The Richardson iteration produces updates of the form

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^0 + q_R^k(\boldsymbol{A})\boldsymbol{r}^0$$

where $q_R^k(\boldsymbol{A})$ is a polynomial of order k given recursively by

$$q_R^k(\boldsymbol{A}) = \boldsymbol{I} + (\boldsymbol{I} - \boldsymbol{A})q_R^{k-1}(\boldsymbol{A})$$

with $q^0(\boldsymbol{A}) = \boldsymbol{I}$, and where $\boldsymbol{r}^0 = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^0$ is the initial residual. Considering the Richardson iteration as a polynomial method highlights the weakness of the method: the Richardson iteration always generates the same sequence of polynomials, regardless of the particular problem to which the method is applied. One implication of this is that, generically, the iteration will not terminate in a finite number of iterations. However, by the Cayley-Hamilton theorem, there exists an n^{th} order polynomial in $\boldsymbol{A}$ which is exactly $\boldsymbol{A}^{-1}$, but in general, the polynomial of order n generated by Richardson iteration will not correspond to the Cayley-Hamilton polynomial. One way of considering conjugate direction methods is that they are methods which at each iteration generate an optimal polynomial for calculating $\boldsymbol{x}^{k+1}$ (optimal in the sense that $\boldsymbol{x}^{k+1}$ minimizes a pre-defined cost functional).

As an example, consider the conjugate gradient (CG) method [7], used to solve (2.1) for the case of symmetric and positive-definite $\boldsymbol{A}$. This method again generates $\boldsymbol{x}^{k+1}$ with polynomials of $\boldsymbol{A}$:

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^0 + q_{CG}^k(\boldsymbol{A})\boldsymbol{r}^0$$

but does so by seeking to minimize the cost functional

$$\phi(\boldsymbol{x}) = \langle \boldsymbol{x}, \boldsymbol{b} - \frac{1}{2}\boldsymbol{A}\boldsymbol{x}\rangle.$$

Here, the inner product $< \boldsymbol{x}, \boldsymbol{y} >$ is the standard Euclidean inner product on $\mathbb{R}^n$. The relation $\boldsymbol{x}^k = \boldsymbol{x}^0 + q_{CG}^{k-1}(\boldsymbol{A})\boldsymbol{r}^0$ implies that $\boldsymbol{x}^k \in \boldsymbol{x}^0 + \mathbb{K}^k(\boldsymbol{r}^0, \boldsymbol{A})$, where $\mathbb{K}^k(\boldsymbol{r}^0, \boldsymbol{A})$ is the k-dimensional Krylov space:

$$\mathbb{K}^k(\boldsymbol{r}^0, \boldsymbol{A}) = \text{span}\{\boldsymbol{r}^0, \boldsymbol{A}\boldsymbol{r}^0, \ldots, \boldsymbol{A}^{k-1}\boldsymbol{r}^0\}.$$

The minimization of ϕ can be accomplished for each iteration k by enforcing the Galerkin condition that the gradient of ϕ be zero on $\mathbb{K}^k(\boldsymbol{r}^0, \boldsymbol{A})$, i.e.,

$$\langle \nabla\phi(\boldsymbol{x}^k), \boldsymbol{y}\rangle = 0 \quad \forall \boldsymbol{y} \in \mathbb{K}^k(\boldsymbol{r}^0, \boldsymbol{A}).$$

It is sufficient to enforce the Galerkin condition on any basis of $\mathbb{K}^k(\boldsymbol{r}^0, \boldsymbol{A})$ that might be chosen. In particular, by choosing $\{\boldsymbol{p}^0, \ldots \boldsymbol{p}^m\}$ as a basis for $\mathbb{K}^{m+1}(\boldsymbol{r}^0, \boldsymbol{A})$, such that

$$\langle \boldsymbol{A}\boldsymbol{p}^i, \boldsymbol{p}^j\rangle = 0 \quad i \neq j,$$

and by using the update

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \frac{\langle \boldsymbol{r}^k, \boldsymbol{p}^k\rangle}{\langle \boldsymbol{A}\boldsymbol{p}^k, \boldsymbol{p}^k\rangle}\boldsymbol{p}^k,$$

the sequence $\{\boldsymbol{x}^1, \boldsymbol{x}^2, \ldots\}$ can be generated iteratively so that $\boldsymbol{x}^{k+1}$ minimizes ϕ on $\mathbb{K}^{k+1}(\boldsymbol{r}^0, \boldsymbol{A})$ for each $k = 0, \ldots, m$ (see [8, pp. 271–273]).

Since the largest amount of work in the CG iteration is in the matrix-vector product, the CG algorithm requires only a modest increase in work per iteration when compared to Richardson-based iteration methods. However, the optimality of the CG algorithm provides a guarantee of finite termination (by Cayley-Hamilton) plus much better convergence properties prior to termination [9]. In fact, the convergence rate of the CG algorithm is bounded by:

$$\|\boldsymbol{e}^k\|_A \leq 2\left(\frac{\sqrt{\kappa(\boldsymbol{A})}-1}{\sqrt{\kappa(\boldsymbol{A})}+1}\right)^k \|\boldsymbol{e}^0\|_A \tag{2.2}$$

where $\|\boldsymbol{e}\|_A = \langle \boldsymbol{A}\boldsymbol{e}, \boldsymbol{e}\rangle^{\frac{1}{2}}$ is the A-norm of $\boldsymbol{e}$ and $\kappa(\boldsymbol{A})$ is the condition number of the matrix $\boldsymbol{A}$. In practice, the bounds given in (2.2) are not necessarily sharp, particularly when $\boldsymbol{A}$ has clustered eigenvalues.

For non-symmetric matrices, the CG algorithm cannot be directly applied. Krylov-space methods which are appropriate for non-symmetric systems include CG applied to the normal equations (CGNR) [7], generalized conjugate residual (GCR) algorithm [10], the generalized minimum residual (GMRES) algorithm [11], and the conjugate gradient squared (CGS) algorithm [12]. These methods are quite powerful and are widely used, but none completely preserves the elegance of the original CG algorithm (see [13] for a discussion of necessary and sufficient conditions for the existence of a conjugate gradient method).

The CGNR algorithm solves (2.1) for non-symmetric $\boldsymbol{A}$ by applying CG to the equivalent symmetric system

$$\boldsymbol{A}^{\dagger}\boldsymbol{A}\boldsymbol{x} = \boldsymbol{A}^{\dagger}\boldsymbol{b},$$

where the superscript $\dagger$ denotes algebraic transposition. However, convergence of CGNR can be drastically slower than convergence of CG. Convergence of CG is bounded by (2.2), so convergence of CGNR is bounded by

$$\|\boldsymbol{e}^k\|_{A^{\dagger}A} = \|\boldsymbol{r}^k\| \leq 2\left(\frac{\sqrt{\kappa(\boldsymbol{A}^{\dagger}\boldsymbol{A})}-1}{\sqrt{\kappa(\boldsymbol{A}^{\dagger}\boldsymbol{A})}+1}\right)^k \|\boldsymbol{r}^0\| = 2\left(\frac{\kappa(\boldsymbol{A})-1}{\kappa(\boldsymbol{A})+1}\right)^k \|\boldsymbol{r}^0\|.$$

For large $\kappa(\boldsymbol{A})$, the convergence of CG is essentially a function of $\sqrt{\kappa(\boldsymbol{A})}$, whereas convergence of CGNR is essentially a function of $\kappa(\boldsymbol{A})$.

The GCR and GMRES algorithms are theoretically equivalent algorithms which seek to minimize $\|\boldsymbol{r}\|^2$ at each iteration. To do this, the basis for the Krylov space must be formed explicitly with an orthogonalization scheme at each iteration so that the work at each iteration grows linearly with the iteration number. Restarted and incomplete orthogonalization versions of these methods seek to bound the length of the orthogonalization process, but in so doing also tend to corrupt the effectiveness of the full orthogonalization versions of the algorithms.

The CGS algorithm [12] uses a low-order recurrence relation at each iteration and abandons guaranteed minimization properties altogether. This is a theoretical drawback, but in practice, CGS seems to work quite reliably.

2.3 Nonlinear Solution Methods

In this section, some methods are discussed for solving systems of nonlinear equations

$$\boldsymbol{F}(\boldsymbol{x}) = 0 \tag{2.3}$$

where $\boldsymbol{x} \in \mathbb{R}^n$ and $\boldsymbol{F} : \mathbb{R}^n \rightarrow \mathbb{R}^n$.

2.3.1 Newton's Method

The most popular method for solving (2.3) is undoubtedly Newton's method, where the n-dimensional linear system of equations

$$\boldsymbol{J}_F(\boldsymbol{x}^m)\boldsymbol{x}^{m+1} = \boldsymbol{J}_F(\boldsymbol{x}^m)\boldsymbol{x}^m - \boldsymbol{F}(\boldsymbol{x}^m)$$

is solved for $\boldsymbol{x}^{m+1}$ in an iterative loop. Here, $\boldsymbol{J}_F(\boldsymbol{x}) = \frac{\partial \boldsymbol{F}(\boldsymbol{x})}{\partial \boldsymbol{x}}$ is the Jacobian of F. Typically, the iteration is performed in two steps:

$$\begin{aligned} \boldsymbol{J}_F(\boldsymbol{x}^m)\Delta\boldsymbol{x}^m &= -\boldsymbol{F}(\boldsymbol{x}^m) \\ \boldsymbol{x}^{m+1} &= \boldsymbol{x}^m + \Delta\boldsymbol{x}^m. \end{aligned} \tag{2.4}$$

Newton's method converges quadratically, provided the initial guess, $\boldsymbol{x}^0$, is sufficiently close to the exact solution [8].

The standard Newton method has some drawbacks. First, a linear system solution step is required at each iteration. This can be expensive in terms of computation and in terms of storage, especially if a direct factorization method is used. Second, global convergence can be problematic if the initial guess is not close enough to the exact solution. Alternative nonlinear solution methods seek to improve the computation, storage, and/or convergence properties of the standard Newton method.

2.3.2 Nonlinear Relaxation

As an alternative to the standard Newton method, (2.3) can be decomposed into smaller sub-problems, each of which is solved independently in an iterative loop, using fixed values from previous iterations for the variables from other sub-problems. Two common decompositions produce the Jacobi-Newton and Seidel-Newton algorithms (see [8]). These methods can be considered to be generalizations of the corresponding linear Gauss-Jacobi and Gauss-Seidel relaxation methods.

The Jacobi-Newton algorithm solves (2.3) by solving the equations

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

for x_i^{m+1}, usually with a scalar form of Newton's method. Similarly, the Seidel-Newton algorithm solves (2.3) by solving the equations

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

for x_i^{m+1}. The essential difference between Jacobi-Newton and Seidel-Newton is that, when computing x_i^{m+1} Seidel-Newton uses the values of x_j^{m+1} for the j^{th} subsystem if it has already been computed, otherwise x_j^m is used. In some sense, Seidel-Newton uses the most recently computed information that is available. The class of nonlinear solution algorithms to which Jacobi-Newton and Seidel-Newton belong are referred to as relaxation-Newton algorithms.

Each iteration of a relaxation-Newton method requires solving a scalar nonlinear problem to determine x_i^{m+1} — usually with a scalar form of Newton's method. Even if the scalar nonlinear solution method is iterated until convergence, the outer loop will generally not be converged for $\boldsymbol{x}$. This suggests that in the early iterations, x_i^{m+1} does not need to be determined very precisely. The n-step relaxation-Newton methods take a predetermined fixed number, n, of scalar Newton iterations — often as few as one [8].

2.3.3 Inexact Newton Methods

The class of nonlinear solution methods known as inexact Newton methods are obtained by combining Newton's method with a linear solution method that only solves the linear system approximately. As in [14], the linear system that is solved with an inexact Newton method can be specified as

$$\begin{aligned} \boldsymbol{J_F}(\boldsymbol{x}^m)\boldsymbol{\Delta x}^m &= -\boldsymbol{F}(\boldsymbol{x}^m) + \boldsymbol{r}^m \\ \boldsymbol{x}^{m+1} &= \boldsymbol{x}^m + \boldsymbol{\Delta x}^m \end{aligned}$$

where $\boldsymbol{r}^m$ is the residual and represents the difference between $\boldsymbol{J_F}(\boldsymbol{x}^m)\boldsymbol{\Delta x}^m$ and $\boldsymbol{F}(\boldsymbol{x}^m)$.

One common context in which inexact Newton methods arise is when an iterative linear solver is used to solve (2.4). In this case, the linear system is only solved approximately to within the convergence criterion of the particular iterative method. Here, methods combining Newton with linear relaxation are referred to as Newton-relaxation methods; methods combining Newton with a conjugate direction method are referred to as Newton-Krylov methods.

A typical Newton-relaxation or Newton-Krylov method has the form shown in Algorithm 2.3.1. Using this formulation, convergence of the iterative linear solution is determined by $\|\boldsymbol{r}^k\|/\|\boldsymbol{r}^0\|$, i.e., the ratio of the linear residual at iteration k to the initial linear residual. As discussed in [14], choosing a fixed convergence criterion $0 < \epsilon < 1$ for all m will result in linear convergence of the nonlinear iteration. However, by scheduling a sequence $\{\epsilon^0, \epsilon^1, \ldots\}$ so that $\epsilon^m < 1$ for all m and $\epsilon^m \to 0$ as $m \to \infty$, the nonlinear iteration will converge superlinearly.

```
Algorithm 2.3.1 (Inexact Newton for solving $F(x) = 0$).

Choose $x^0$, $\epsilon^0$.
For $m = 0, 1, \ldots$ until converged
    Pick $\Delta x^{m,0}$
    Set $r^0 = -F(x^m) - J_F(x^m)\Delta x^{m,0}$
    For $k = 0, 1, \ldots$ until $\|r^k\|/\|r^0\| < \epsilon^m$
        Perform relaxation or Krylov linear iteration steps
    Set $x^{m+1} = x^m + \Delta x^{m,k}$
```

2.3.4 Matrix-Free Methods

One modification that can be made to Newton-relaxation or Newton-Krylov methods is to dispense with the explicit formation of the Jacobian. The iterative linear solvers require only the result of a matrix-vector product, not the matrix itself. Since the matrix for the linear system in question is the Jacobian of a nonlinear function $\boldsymbol{F}$, an approximate matrix-vector product can be calculated according to

$$J_F(\boldsymbol{x})\boldsymbol{p} = \frac{\partial \boldsymbol{F}(\boldsymbol{x})}{\partial \boldsymbol{x}}\boldsymbol{p} \approx \frac{1}{\sigma}\left[\boldsymbol{F}(\boldsymbol{x} + \sigma \boldsymbol{p}) - \boldsymbol{F}(\boldsymbol{x})\right]$$

where σ is a small scalar parameter.

The use of matrix-free Newton-Krylov methods within the context of solving stiff systems of ODE's was first studied by Gear and Saad in [15] and subsequently studied by Brown and Hindmarsh [16, 17] and Chan and Jackson [18]. Matrix-free Newton-Krylov methods with global convergence properties are examined in [19].

2.4 Integration Methods

Many initial value problems admit an explicit representation as

$$\begin{aligned} \tfrac{d}{dt}\boldsymbol{x}(t) &= \boldsymbol{F}(\boldsymbol{x}(t), t) \\ \boldsymbol{x}(0) &= \boldsymbol{x}_0 \end{aligned} \tag{2.5}$$

where $\boldsymbol{x}(t) \in \mathbb{R}^N$ and $\boldsymbol{F} : \mathbb{R}^{N+1} \to \mathbb{R}^N$.

A linear multistep integration formula applied to solving (2.5) is expressed by:

$$\sum_{i=0}^{s} \alpha_i \boldsymbol{x}(t_{m+1-i}) = h_m \sum_{i=0}^{s} \beta_i \boldsymbol{F}(\boldsymbol{x}(t_{m+1-i}), t_{m+1-s})$$

where $h_m = t_{m+1} - t_m$ is the discretization timestep, $\boldsymbol{x}(t_m)$ is the estimated value of the solution at time $t = t_m$, $\boldsymbol{F}(\boldsymbol{x}(t), t)$ represents the dynamics of the equation at the given timepoint using the estimated value, the parameters α_i and β_i are chosen for accuracy within stability limits, and s is the order of the formula [20, 21]. If $\beta_0 = 0$, $\boldsymbol{x}(t_{m+1})$ can be found explicitly as a function of previous values of $\boldsymbol{x}$ and $\boldsymbol{F}(\cdot)$ evaluated at previous values of $\boldsymbol{x}$. For $\beta_0 \neq 0$, an implicit equation in terms of $\boldsymbol{x}(t_{m+1})$ and $\boldsymbol{F}(\boldsymbol{x}(t_{m+1}), t_{m+1})$ must be solved for $\boldsymbol{x}(t_{m+1})$. Hence, one obtains the denotation "explicit method" for $\beta_0 = 0$ and "implicit method" for $\beta_0 \neq 0$.

For problems such as (2.5), explicit methods have a potentially significant advantage over implicit methods because there is no need to perform a nonlinear system solution. Without a nonlinear system solution step, the system Jacobian and linear system solution are obviated, resulting in a substantial reduction in computation as well as storage. However, explicit methods are far less stable than their implicit counterparts. For problems having eigenvalues that differ by several orders of magnitude (i.e., stiff problems), implicit methods are computationally superior to explicit methods. The stability of the implicit methods allows for substantially larger timesteps, resulting in lower overall computational work for the simulation.

Explicit integration methods also lose their advantages when the differential portion of the initial value problem is itself implicit. e.g., the circuit transient simulation problem

$$\begin{aligned} \tfrac{d}{dt}\boldsymbol{q}\left(\boldsymbol{v}(t), t\right) + \boldsymbol{i}\left(\boldsymbol{v}(t), t\right) &= 0 \\ \boldsymbol{v}(0) &= \boldsymbol{v}_0. \end{aligned} \tag{2.6}$$

A linear multistep integration formula applied to solving (2.6) is expressed by:

$$\sum_{i=0}^{s} \alpha_i \boldsymbol{q}(\boldsymbol{v}(t_{m+1-i})) = h_m \sum_{i=0}^{s} \beta_i \boldsymbol{i}(\boldsymbol{v}(t_{m+1-i}), t_{m+1-s}) \tag{2.7}$$

Even if an explicit integration method is used, i.e., by choosing $\beta_0 = 0$, (2.7) is still implicit in $\boldsymbol{v}(t_{m+1})$. Therefore, a nonlinear solution step is still required, but the stability inherent to an implicit integration method is not retained. In special cases, some advantage can be gained because $\boldsymbol{q}$ may be easier to invert than $\boldsymbol{i}$, but in general this advantage is not enough to compensate for the lack of the strong stability properties of implicit integration methods.

2.5 Waveform Methods

The discussion in the previous sections concentrated on different methods that could replace the three components of the "implicit-integration, Newton, direct method" approach. Another means of obtaining a computational advantage in solving (1.1) is in selecting an alternative decomposition of the original problem. It has already been shown how alternative decompositions applied to the nonlinear and linear system solution steps produced nonlinear and linear relaxation algorithms. If this type of decomposition is applied at the ODE level, one obtains waveform methods.

One approach to studying waveform methods is to consider the IVP to be an operator equation on some function space. The traditional waveform methods can then be derived as extensions of nonlinear solution methods to that space. It is this point of view which leads to the derivation of the waveform conjugate direction methods in Chapter 4, for instance.

The operator formulation of the IVP can be written as

$$\mathcal{F}\boldsymbol{x} = 0 \tag{2.8}$$

where $\boldsymbol{x}$ is now a member of some function space and $\mathcal{F}$ is a nonlinear differential operator. An example definition of $\mathcal{F}$ is

$$\mathcal{F}\boldsymbol{x}(t) = \tfrac{d}{dt}\boldsymbol{x}(t) + \boldsymbol{F}(\boldsymbol{x}(t), t) \tag{2.9}$$

with $\boldsymbol{x} \in \mathbb{C}^1(\boldsymbol{x}_0, [0,T], \mathbb{R}^n)$, $\mathcal{F} : \mathbb{C}^1(\boldsymbol{x}_0, [0,T], \mathbb{R}^n) \rightarrow \mathbb{C}^1(\boldsymbol{x}_0, [0,T], \mathbb{R}^n)$ and the function space $\mathbb{C}^1(\boldsymbol{x}_0, [0,T], \mathbb{R}^n)$ defined as

$$\mathbb{C}^1(\boldsymbol{x}_0, [0,T], \mathbb{R}^n) = \{\boldsymbol{f} \in \mathbb{C}^1([0,T], \mathbb{R}^n) | \boldsymbol{f}(0) = \boldsymbol{x}_0\}.$$

Note that $\mathcal{F}$ is at best densely defined on $\mathbb{C}^1(\boldsymbol{x}_0, [0,T], \mathbb{R}^n)$.

Given the formulation of the IVP in (2.8) one can apply abstracted forms of the methods described Section 2.3 to obtain various nonlinear waveform methods. These abstracted nonlinear methods are discussed in the following sections.

2.5.1 Waveform Newton Methods

The waveform Newton method is obtained by applying an abstracted form of Newton's method to (2.8). This method is discussed in [22] and applied to the circuit simulation problem in [23] and [24]. The waveform Newton method is expressed as

$$\begin{aligned} \mathcal{J}_{\mathcal{F}}(\boldsymbol{x}^m)\boldsymbol{\Delta x}^m &= -\mathcal{F}(\boldsymbol{x}^m) \\ \boldsymbol{x}^{m+1} &= \boldsymbol{x}^m + \boldsymbol{\Delta x}^m. \end{aligned}$$

where $\mathcal{J}_{\mathcal{F}}$ is the Frechet derivative of $\mathcal{F}$ defined by

$$\mathcal{J}_{\mathcal{F}}\boldsymbol{x}(t) = \tfrac{d}{dt}\boldsymbol{x}(t) + \boldsymbol{J}_F(\boldsymbol{x}(t), t). \tag{2.10}$$

Using (2.9) and (2.10), the waveform Newton method can be expressed as

$$\begin{aligned}\left(\tfrac{d}{dt} + \boldsymbol{J_F}(\boldsymbol{x}^m(t),t)\right)\boldsymbol{\Delta x}^m(t) &= -\tfrac{d}{dt}\boldsymbol{x}^m(t) - \boldsymbol{F}(\boldsymbol{x}^m(t),t)\\ \boldsymbol{x}^{m+1}(0) &= \boldsymbol{x}_0\end{aligned}$$

which can be rearranged to remove the $\frac{d}{dt}$ from the right-hand side to obtain the linear IVP

$$\begin{aligned}\left(\tfrac{d}{dt} + \boldsymbol{J_F}(\boldsymbol{x}^m)\right)\boldsymbol{x}^{m+1} &= \boldsymbol{J_F}(\boldsymbol{x}^m)\boldsymbol{x}^m - \boldsymbol{F}(\boldsymbol{x}^m)\\ \boldsymbol{x}^{m+1}(0) &= \boldsymbol{x}_0.\end{aligned}$$

This linear IVP can be solved with a variety of methods, as can be seen in Figure 2-1. For instance, the problem can be immediately discretized and a linear system solver can be used to solve the resulting sequence of matrix problems[2]. Alternatively, the linear IVP can be solved iteratively with a linear waveform relaxation method or with a conjugate direction waveform method. In this case, the problem is discretized (and the resulting linear systems solved) within the main iterative loop.

A discussion of the convergence properties of the waveform Newton method can be found in [23].

2.5.2 Waveform Relaxation Methods

As with the linear and nonlinear relaxation methods, the nonlinear IVP can be decomposed in space and solved iteratively. The Jacobi waveform relaxation algorithm solves (1.1) by solving the scalar IVP's

$$\begin{aligned}\tfrac{d}{dt}x_i^k(t) + f_i(x_1^k(t),\ldots,x_{i-1}^k(t),x_i^{k+1}(t),x_{i+1}^k(t),\ldots,x_n^k(t),t) &= 0\\ x_i^k(0) &= x_{0_i}\end{aligned}$$

for each $x_i^{k+1}(t)$ with a scalar integration scheme. The historical basis for waveform relaxation methods is the Picard-Lindelöf iteration, used to demonstrate existence and uniqueness of solutions to IVP's [25]. Waveform relaxation has been used very successfully for simulating VLSI circuits [26, 27]. Convergence theory for the linear time-invariant case is studied in [28] for ODE's and in [29] for DAE's.

2.5.3 Inexact Waveform Newton Methods

One can continue to maintain the analogy between nonlinear solution methods and waveform methods and construct the class of inexact waveform Newton methods. These methods would result from the combination of the waveform Newton method and an iterative

[2]Or, as in [24], an n-dimensional linear IVP discretized with m timepoints can be treated as one $mn \times mn$ problem instead of a sequence of m separate $n \times n$ systems.

linear waveform method. The combination of waveform Newton with a linear waveform relaxation method is sometimes known as waveform Newton relaxation (WNR) and is discussed in [30]. The combination of waveform Newton with a linear waveform conjugate direction method is presented for the first time in [31] and developed more fully in Chapter 4 of this thesis.

2.6 Parallel Techniques

In addition to improving the algorithms for solving (1.1), significant computational time-savings can be gained by using parallel processing hardware. However, no true general-purpose parallel machine yet exists — obtaining good parallel performance requires careful matching of problem, algorithm, and architecture.

Solving IVP's in parallel is difficult because the initial value problem is inherently serial due to the ODE structure. The approaches used for parallelizing IVP solution algorithms are inherently tied to the original decomposition of the problem. For instance, with a point-wise decomposition, one would attempt to parallelize the solution steps required at every timepoint, since those solutions are generated sequentially. On the other hand, for a waveform-based solution, one would attempt to parallelize the solution steps required for the set of waveforms.

In general, iterative techniques are easier to parallelize than direct techniques. For instance, the waveform and point relaxation methods solve a sequence of equations for x_i^{k+1} using values of other components of x from previous iterations. An obvious use of parallel hardware for such methods is to assign each component of x to a separate processor. Each processor i is then responsible for obtaining those values of x from other processors that are necessary to calculate x_i^{k+1}.

Relaxation methods can also use a technique known as chaotic relaxation in which the processors run asynchronously [32], i.e., each processor has a local iteration count independent of the iteration count of the other processors. Each processor computes the value of its assigned variable using whatever values for the components from other processors are available at the time. The conjugate direction methods, however, require a global control thread and all processors must synchronize several times during each iteration. Whether the superior convergence properties of the conjugate direction methods will outweigh the added synchronization cost in a parallel implementation is an open question.

Many methods have been developed for parallelizing direct techniques, and a very good survey of such methods can be found in [33]. However, the successful techniques rely on certain special structures of the problems to be solved (such as the matrix possessing a band structure). There has not been a truly successful parallel implementation of sparse Gaussian elimination for the types of matrices produced by the general circuit simulation problem, for instance. There is some recent interesting work in this direction by Karmarkar, however [34, 35].

References

[1] G. Strang, *Linear Algebra and Its Applications.* New York: Academic Press, 1980.

[2] G. H. Golub and C. F. Van Loan, *Matrix Computations.* Baltimore, Maryland: The John Hopkins University Press, 1983.

[3] I. S. Duff, A. M. Erisman, and J. K. Reid, *Direct Methods for Sparse Matrices.* Oxford: Clarendon Press, 1986.

[4] V. Strassen, "Gaussian elimination is not optimal," *Numer. Math.*, vol. 13, pp. 354–356, 1968.

[5] L. F. Richardson, "The approximate arithmetical solution by finite differences of physical problems involving differential equations,with applications to the stress in a masonry dam," *Philos. Trans. Roy. Soc. London*, vol. Ser. A210, pp. 307–357, 1910.

[6] R. S. Varga, *Matrix Iterative Analysis.* Automatic Computation Series, Englewood Cliffs, New Jersey: Prentice-Hall Inc, 1962.

[7] M. R. Hestenes and E. Stiefel, "Methods of conjugate gradients for solving linear systems," *Journal of Research of the National Bureau of Standards*, vol. 49, pp. 409–436, December 1952.

[8] J. M. Ortega and W. C. Rheinbolt, *Iterative Solution of Nonlinear Equations in Several Variables.* Computer Science and Applied Mathematics, New York: Academic Press, 1970.

[9] O. Axelsson, "Solution of linear systems of equations: Iterative methods," in *Sparse Matrix Techniques* (V. A. Barker, ed.), pp. 1–51, New York: Springer-Verlag, 1976.

[10] H. C. Elman, *Iterative Methods for Large Sparse Nonsymmetric Systems of Linear Equations.* PhD thesis, Computer Science Dept., Yale University, New Haven, CT, 1982.

[11] Y. Saad and M. Schultz, "GMRES: A generalized minimum residual algorithm for solving nonsymmetric linear systems," *SIAM J. Sci. Statist. Comput.*, vol. 7, pp. 856–869, July 1986.

[12] P. Sonneveld, "CGS, a fast Lanczos-type solver for nonsymmetric linear systems," *SIAM J. Sci. Statist. Comput.*, vol. 10, pp. 36–52, 1989.

[13] V. Faber and T. Manteuffel, "Necessary and sufficient conditions for the existence of a conjugate gradient method," *SIAM J. Numer. Anal.*, vol. 21, pp. 352–362, 1984.

[14] R. S. Dembo, S. C. Eisenstat, and T. Steihaug, "Inexact Newton methods," *SIAM J. Numer. Anal.*, vol. 19, pp. 400–408, April 1982.

[15] C. W. Gear and Y. Saad, "Iterative solution of linear equations in ODE codes," *SIAM J. Sci. Statist. Comput.*, vol. 4, pp. 583–601, December 1983.

[16] P. N. Brown and A. C. Hindmarsh, "Matrix-free methods for stiff systems of ODE's," *SIAM J. Numer. Anal.*, vol. 23, pp. 610–638, June 1986.

[17] P. N. Brown and A. C. Hindmarsh, "Reduced storage methods in stiff ODE systems," *J. Appl. Math. Comput.*, vol. 31, pp. 40–91, 1989.

[18] T. F. Chan and K. R. Jackson, "The use of iterative linear equation solvers in codes for large systems of stiff IVP's for ODE's," *SIAM J. Sci. Statist. Comput.*, vol. 7, pp. 378–417, 1986.

[19] P. Brown and Y. Saad, "Hybrid Krylov methods for nonlinear systems of equations," *SIAM J. Sci. Statist. Comput.*, vol. 11, pp. 450–481, May 1990.

[20] C. W. Gear, *Numerical Initial Value Problems in Ordinary Differential Equations.* Automatic Computation, Englewood Cliffs, New Jersey: Prentice-Hall, 1971.

[21] G. Dahlquist and Å. Björck, *Numerical Methods.* Automatic Computation, Englewood Cliffs, New Jersey: Prentice-Hall, 1974.

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

[23] R. Saleh and J. White, "Accelerating relaxation algorithms for circuit simulation using waveform-newton and step-size refinement," *IEEE Trans. CAD*, vol. 9, no. 9, pp. 951–958, 1990.

[24] L. M. Silveira, "Circuit simulation algorithms for massively parallel processors," Master's thesis, Massachusetts Institute of Technology, May 1990.

[25] M. Vidyasagar, *Nonlinear Systems Analysis.* Englewood Cliffs, NJ: Prentice-Hall, 1978.

[26] E. Lelarasmee, A. E. Ruehli, and A. L. Sangiovanni-Vincentelli, "The waveform relaxation method for time domain analysis of large scale integrated circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 1, pp. 131–145, July 1982.

[27] J. K. White and A. Sangiovanni-Vincentelli, *Relaxation Techniques for the Simulation of VLSI Circuits.* Engineering and Computer Science Series, Norwell, Massachusetts: Kluwer Academic Publishers, 1986.

[28] U. Miekkala and O. Nevanlinna, "Convergence of dynamic iteration methods for initial value problems," *SIAM J. Sci. Stat. Comp.*, vol. 8, pp. 459–467, 1987.

[29] U. Miekkala, "Dynamic iteration methods applied to linear DAE systems," *J. Comput. Appl. Math.*, vol. 25, pp. 133–151, 1989.

[30] E. Z. Xia, "Parallel waveform-relaxation-newton for circuit simulation," Master's thesis, University of Illinois at Urbana-Champaign, 305 Talbot, Urbana-Champaign, IL 61801-2932, 1988. also University of Illinois Center for Supercomputing Research and Development Report No. 772.

[31] A. Lumsdaine, M. Reichelt, and J. White, "Conjugate direction waveform methods for transient two-dimensional simulation of MOS devices," in *International Conference on Computer Aided-Design*, (Santa Clara, California), pp. 116–119, November 1991.

[32] A. Chazan and W. Miranker, "Chaotic relaxation," *Linear Algebra and its Applications*, vol. 2, 1969.

[33] K. A. Gallivan *et al*, *Parallel Algorithms for Matrix Computations.* Philadelphia: SIAM, 1990.

[34] N. K. Karmarkar, "A new parallel architecture for sparse matrix computation based on finite projective geometries," *Proceedings of Supercomputing Symposium '91*, June 1991.

[35] I. S. Dhillon, N. K. Karmarkar, and K. G. Ramakrishnan, "An overview of the compilation process for a new parallel architecture," 1991. Unpublished.

3

Parallel Simulation Algorithms for Grid-Based Analog Signal Processors

3.1 Introduction

The recent success in using one- and two-dimensional resistive grids to perform certain filtering tasks required for early vision [1] has sparked interest in general analog signal processors based on arrays of analog circuits coupled by resistive grids. As is usually the case, before fabricating these analog signal processors, substantial circuit-level simulation must be performed to insure correct functionality. Although desirable, simulation of these types of signal processors is particularly difficult because they must be simulated in their entirety at the analog level. Ambitious circuits consist of arrays of cells where the array size can be as large as 256×256, and each cell may contain up to a few dozen devices [2]. Therefore, simulation of a complete signal processor requires solving a system of differential equations with *hundreds of thousands* of unknowns.

The structure of grid-based analog signal processors is such that they can be simulated quickly and accurately with specialized algorithms tuned to certain parallel computer architectures. In particular, the coupling between cells in the analog array is such that a block-iterative scheme can be used to solve the equations generated by an implicit time-discretization scheme, and furthermore, the regular structure of the problem implies that the simulation computations can be accelerated by a massively parallel SIMD computer, such as the Connection Machine®[3].

Connection Machine is a registered trademark of Thinking Machines Corporation.

In this chapter, algorithms are presented for simulating grid-based analog signal processors on a massively parallel computer. In Section 3.2, motivation is provided for this work by way of an idealized example of a grid-based analog signal processor, and a general model of grid-based circuits is developed. The simulation algorithms for performing transient simulation of grid-based circuits are discussed in Section 3.3 and the massively parallel implementation of the algorithms is presented in Section 3.4. Experimental results using the Connection Machine are presented in Section 3.5. Finally, conclusions and suggestions for further work are contained in Section 3.6.

3.2 Problem Description

Preceding the discussion of the simulation algorithms, a description of the problem to be solved is presented in order to highlight the salient features of the problem which will be exploited later by the algorithms. An idealized grid-based analog signal processor is presented first as a motivational problem and then a general description for these types of circuits is developed.

3.2.1 Motivational Problem

Consider the circuit in Figure 3-1, an idealized version of a grid-based analog signal processor used for two-dimensional image smoothing and segmentation [4]. The Kirchoff's current law (KCL) equation for a node at grid location (j,k) in the network is

$$\begin{aligned} c\dot{v}_{j,k} &= g_f(v_{j,k} - u_{j,k}) \\ &+ g_s(v_{j,k} - v_{j+1,k}) + g_s(v_{j,k} - v_{j-1,k}) \\ &+ g_s(v_{j,k} - v_{j,k+1}) + g_s(v_{j,k} - v_{j,k-1}), \end{aligned}$$

where $u_{j,k}$ and $v_{j,k}$ represent the input and processed output image data at the grid point (j,k), respectively, g_f is the input source impedance, c is the parasitic capacitance from the grid node to ground, and $g_s(\cdot)$ is a nonlinear "fused" resistor. In this circuit, the g_s resistors pass currents in such a way as to force $v_{j,k}$ to be a spatially smoothed version of $u_{j,k}$, unless the difference between neighboring $u_{j,k}$'s is very large. In that case, g_s no longer conducts, there is no smoothing, and the image is said to be "segmented" at that point.

In a physical implementation of the image smoothing and segmentation circuit, the idealized elements in the circuit in Figure 3-1 are replaced by subcircuits of physical circuit elements. For example, in Mead's Silicon Retina [1], the voltage source $u_{j,k}$ and

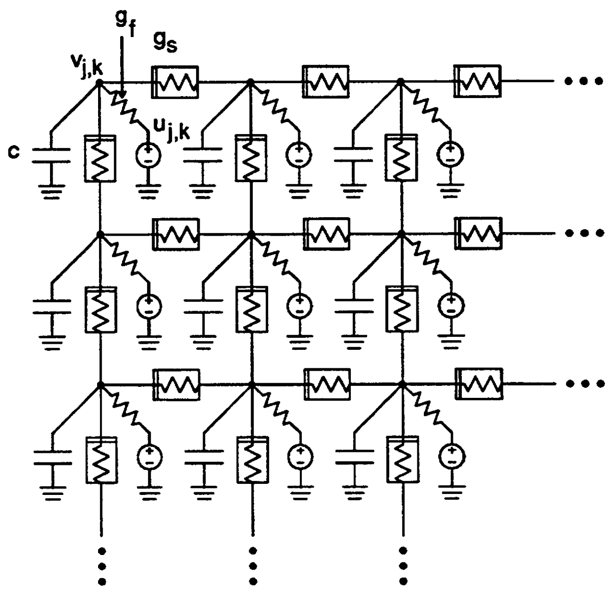

Figure 3-1: Grid of nonlinear resistors.

the source admittance g_f are replaced with a subcircuit containing transconductance amplifiers and a phototransistor; the g_s nonlinear resistor is replaced with a subcircuit comprised of biasing circuitry and MOS transistors (see Section 3.5).

3.2.2 General Array Description

The circuit grid can be represented generally as an $N \times N$ array of identical subcircuits, each of which is connected to its four nearest neighbors. Consider a single such subcircuit, shown abstractly as a multiterminal element in Figure 3-2. Let the subcircuit have M_{int} internal nodes and let it be connected to its nearest neighbor to the north, east, west, and south with M_N, M_E, M_W, and M_S terminals, respectively. For present purposes, it is assumed that the subcircuit acts as a voltage-controlled element with respect to its terminals.

In order to create an $N \times N$ grid circuit of subcircuits, a single subcircuit must be replicated N^2 times, and then each subcircuit must be connected to its four nearest neighbors. The following proposition and its corollaries are provided as a means of describing the grid circuit behavior, given the description of the behavior of the individual subcircuits.

Proposition 3.2.1. Let $\tilde{C}$ be an $n+m$ node circuit with nodal voltage vector $\tilde{v}(t) \in \mathbb{R}^{n+m}$, and nodal charge and current vectors $\tilde{q}(\tilde{v}(t),t), \tilde{i}(\tilde{v}(t),t) \in \mathbb{R}^{n+m}$, respectively. Consider a second circuit, C, which is formed from $\tilde{C}$ by joining each node $j = n+1,\ldots,n+m$ to some node $k_j \in \{1,\ldots,n\}$, such that C is a well-defined circuit and has

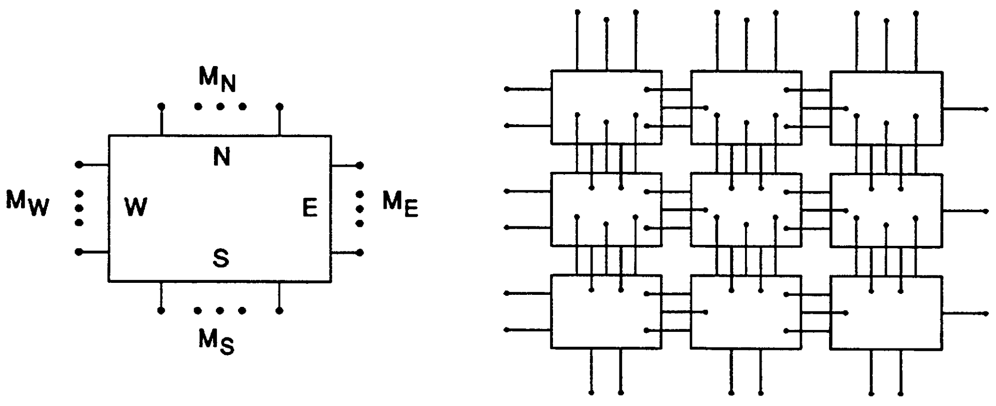

Figure 3-2: A single subcircuit, shown here as a multiterminal element, and a grid constructed of such elements.

nodal voltage vector $\boldsymbol{v}(t) \in \mathbb{R}^n$, and nodal charge and current vectors $\boldsymbol{q}(\boldsymbol{v}(t),t), \boldsymbol{i}(\boldsymbol{v}(t),t) \in \mathbb{R}^n$, respectively. Then, there exists a topological matrix $\boldsymbol{H}$, such that

$$\begin{aligned}\boldsymbol{q}(\boldsymbol{v}(t),t) &= \boldsymbol{H}^\dagger\tilde{\boldsymbol{q}}(\boldsymbol{H}\boldsymbol{v}(t),t)\\ \boldsymbol{i}(\boldsymbol{v}(t),t) &= \boldsymbol{H}^\dagger\tilde{\boldsymbol{i}}(\boldsymbol{H}\boldsymbol{v}(t),t).\end{aligned}$$

Proof. Define $\boldsymbol{H} : \mathbb{R}^n \longrightarrow \mathbb{R}^{n+m}$ by:

$$H_{j,k} = \begin{cases} 1 & \text{if } j = k \\ 1 & \text{if node } j \text{ in } \tilde{\mathcal{C}} \text{ was joined to node } k \text{ in } \mathcal{C} \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

It should be obvious that substituting $\boldsymbol{H}\boldsymbol{v}(t)$ for $\tilde{\boldsymbol{v}}(t)$ will give all devices in $\tilde{\mathcal{C}}$ the same terminal voltages as if the nodes were connected as in $\mathcal{C}$. Thus, each device in $\tilde{\mathcal{C}}$ will produce the same charge at its terminals as it would in $\mathcal{C}$. Consider a component q_j of the vector $\boldsymbol{q}(\boldsymbol{v}(t),t)$, corresponding to the sum of charges at node j in $\mathcal{C}$. Node j will either correspond directly to node j in $\tilde{\mathcal{C}}$ or to node k joined with node j in $\tilde{\mathcal{C}}$. In the former case, $q_j = \tilde{q}_j$, in the latter, $q_j = \tilde{q}_k + \tilde{q}_j$. Therefore, $\boldsymbol{q}(\boldsymbol{v}(t),t) = \boldsymbol{H}^\dagger\tilde{\boldsymbol{q}}(\boldsymbol{H}\boldsymbol{v}(t),t)$. An identical argument shows that $\boldsymbol{i}(\boldsymbol{v}(t),t) = \boldsymbol{H}^\dagger\tilde{\boldsymbol{i}}(\boldsymbol{H}\boldsymbol{v}(t),t)$. □

An alternative proof can be constructed using branch incidence matrices:

Proof. The topological matrix $\boldsymbol{H}$ simply maps the correspondence of node numbers between the nodes in circuit $\tilde{\mathcal{C}}$ and $\mathcal{C}$. Define $\boldsymbol{H} : \mathbb{R}^n \longrightarrow \mathbb{R}^{n+m}$ by:

$$H_{j,k} = \begin{cases} 1 & \text{if } j = k \\ 1 & \text{if node } j \text{ in } \tilde{\mathcal{C}} \text{ was joined to node } k \text{ in } \mathcal{C} \\ 0 & \text{otherwise} \end{cases} \tag{3.2}$$

Let $\tilde{C}$ and C be described abstractly as collections of nodes and edges. Then C will have the same edges as $\tilde{C}$ but fewer nodes. In addition, those edges that meet at nodes in $\tilde{C}$ will also meet at nodes in C. However, some nodes will be merged so that some of the branches in $\tilde{C}$ which meet at different nodes will meet at the same nodes in C. Thus, the branch incidence matrix $\boldsymbol{B}$ for circuit C is related to the branch incidence matrix $\tilde{\boldsymbol{B}}$ for circuit $\tilde{C}$ according to:

$$\boldsymbol{B}^\dagger = \tilde{\boldsymbol{B}}^\dagger \boldsymbol{H},$$

or, equivalently,

$$\boldsymbol{B} = \boldsymbol{H}^\dagger \tilde{\boldsymbol{B}}.$$

The proposition follows by reducing the sparse tableau formulation for voltages and currents for C to the nodal formulation. That is, the nodal current vectors $\boldsymbol{i}$ and $\tilde{\boldsymbol{i}}$ are defined by

$$\begin{aligned} \boldsymbol{i}(\boldsymbol{v}) &= \boldsymbol{B}\boldsymbol{g}(\boldsymbol{B}^\dagger \boldsymbol{v}) \\ \tilde{\boldsymbol{i}}(\tilde{\boldsymbol{v}}) &= \tilde{\boldsymbol{B}}\boldsymbol{g}(\tilde{\boldsymbol{B}}^\dagger \tilde{\boldsymbol{v}}), \end{aligned}$$

where $\boldsymbol{g}$ is the vector of branch currents for $\tilde{C}$ and C, and dependence on t is omitted for clarity. Substituting $\boldsymbol{H}^\dagger \tilde{\boldsymbol{B}}$ for $\boldsymbol{B}$ produces

$$\begin{aligned} \boldsymbol{i}(\boldsymbol{v}) &= \boldsymbol{H}^\dagger \tilde{\boldsymbol{B}}\boldsymbol{g}(\tilde{\boldsymbol{B}}^\dagger \boldsymbol{H}\boldsymbol{v}) \\ &= \boldsymbol{H}^\dagger \tilde{\boldsymbol{i}}(\boldsymbol{H}\boldsymbol{v}). \end{aligned}$$

An identical argument shows that $\boldsymbol{q}(\boldsymbol{v}) = \boldsymbol{H}^\dagger \tilde{\boldsymbol{q}}(\boldsymbol{H}\boldsymbol{v})$. □

Example. Consider the circuit graphs shown in Figure 3-3. In this case the $\boldsymbol{H}$ matrix relating the two circuits is given by:

$$H = \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & & 1 \\ & 1 & & \end{bmatrix}. \tag{3.3}$$

Remark. The matrix $\boldsymbol{H}$ can be defined more generally by relaxing the condition on the ordering of nodes in the hypothesis of Proposition 3.2.1. That is, C can be constructed from $\tilde{C}$ by joining a set of m distinct nodes $\{k_1, \ldots, k_m\} \subset \{1, \ldots, n+m\}$ to some of the n nodes $\{1, \ldots, n+m\} \backslash \{k_1, \ldots, k_m\}$. The remaining n nodes are then renumbered from $\{1, \ldots, n\}$. The matrix $\boldsymbol{H}$ then maps the former node numbers $\{1, \ldots, n+m\}$ to the new node numbers $\{1, \ldots, n\}$.

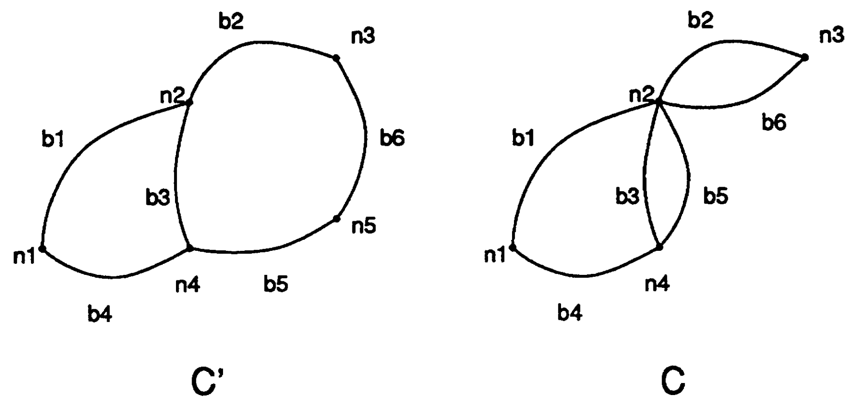

Figure 3-3: Example graphs for circuits $\tilde{C}$ and C. The circuits both have branches b1 through b6; circuit $\tilde{C}$ has nodes n1 through n5, while circuit C has nodes n1 through n4. The topological matrix relating the nodes for this example is given in (3.3).

The following corollary provides a simple way of describing the construction of a grid circuit from subcircuits.

Corollary 3.2.2. Consider an $N \times N$ circuit grid of identical subcircuits, as shown in Figure 3-2. Assume each subcircuit has M_{int} internal nodes plus M_{term} terminals, and define $\tilde{M} = M_{int} + M_{term}$. Let $\tilde{\boldsymbol{v}}_\alpha(t) \in \mathbb{R}^{\tilde{M}}$ be the nodal voltage vector for the subcircuit at grid location α when that subcircuit is separated from the grid, and let $\tilde{\boldsymbol{q}}_\alpha(\tilde{\boldsymbol{v}}_\alpha(t), t) \in \mathbb{R}^{\tilde{M}}$ and $\tilde{\boldsymbol{i}}_\alpha(\tilde{\boldsymbol{v}}_\alpha(t), t) \in \mathbb{R}^{\tilde{M}}$ be the associated nodal charge and current vectors, respectively. Next, define $\tilde{n} = N^2\tilde{M}$, let $\tilde{\boldsymbol{v}} \in \mathbb{R}^{\tilde{n}}$ represent the nodal voltage vector for the N^2 subcircuits separated from each other, and let $\tilde{\boldsymbol{q}}(\tilde{\boldsymbol{v}}(t), t) \in \mathbb{R}^{\tilde{n}}$ and $\tilde{\boldsymbol{i}}(\tilde{\boldsymbol{v}}(t), t) \in \mathbb{R}^{\tilde{n}}$ be the associated nodal charge and current vectors, respectively. Finally, define $n = N^2 M_{int}$, let $\boldsymbol{v}(t) \in \mathbb{R}^n$ represent the voltage vector for the connected grid circuit, and let $\boldsymbol{q}(\boldsymbol{v}(t), t) \in \mathbb{R}^n$ and $\boldsymbol{i}(\boldsymbol{v}(t), t) \in \mathbb{R}^n$ be the associated nodal charge and current vectors, respectively.

Then, $\boldsymbol{q}$ and $\boldsymbol{i}$ can be related to $\tilde{\boldsymbol{q}}$ and $\tilde{\boldsymbol{i}}$ by a topological matrix $\boldsymbol{H} : \mathbb{R}^{\tilde{n}} \to \mathbb{R}^n$, such that

$$\begin{aligned} \boldsymbol{q}(\boldsymbol{v}(t), t) &= \boldsymbol{H}^\dagger \tilde{\boldsymbol{q}}(\boldsymbol{H}\boldsymbol{v}(t), t) \\ \boldsymbol{i}(\boldsymbol{v}(t), t) &= \boldsymbol{H}^\dagger \tilde{\boldsymbol{i}}(\boldsymbol{H}\boldsymbol{v}(t), t). \end{aligned}$$

Proof. Order the subcircuit terminals such that nodes $\{1, \ldots, n\}$ correspond only to the internal nodes of the subcircuits and nodes $\{n+1, \ldots, \tilde{n}\}$ correspond to the terminal nodes of the subcircuits. Apply Proposition 3.2.1. $\square$

Remark. If all N^2 subcircuits are identical, then $\tilde{n} = N^2 M_{int} + N^2 M_{term}$ and $n = N^2 M_{int}$. However, the boundary subcircuits may differ from the subcircuits internal to the grid in such a way that the boundary subcircuits have no unconnected terminals. In that case, the values of $\tilde{n}$ and n will be slightly different from the above, depending on the particular differences of the values M_{int} and M_{term} for the boundary subcircuits.

Corollary 3.2.3. Let $\boldsymbol{J}_q = \frac{\partial \boldsymbol{q}}{\partial \boldsymbol{v}}$, $\boldsymbol{J}_i = \frac{\partial \boldsymbol{i}}{\partial \boldsymbol{v}}$, $\boldsymbol{J}_{\tilde{q}} = \frac{\partial \tilde{\boldsymbol{q}}}{\partial \boldsymbol{v}}$, and $\boldsymbol{J}_{\tilde{i}} = \frac{\partial \tilde{\boldsymbol{i}}}{\partial \boldsymbol{v}}$ be the Jacobian matrices for the functions $\boldsymbol{q}$, $\boldsymbol{i}$, $\tilde{\boldsymbol{q}}$, and $\tilde{\boldsymbol{i}}$ described in Corollary 3.2.2, respectively, and let $\boldsymbol{H}$ be the topological matrix relating $\boldsymbol{q}$ to $\tilde{\boldsymbol{q}}$ and $\boldsymbol{i}$ to $\tilde{\boldsymbol{i}}$. Then, $\boldsymbol{J}_q$ is related to $\boldsymbol{J}_{\tilde{q}}$ and $\boldsymbol{J}_i$ is related to $\boldsymbol{J}_{\tilde{i}}$ by

$$\boldsymbol{J}_q(\boldsymbol{v}(t),t) = \boldsymbol{H}^\dagger \boldsymbol{J}_{\tilde{q}}(\boldsymbol{H}\boldsymbol{v}(t),t)\boldsymbol{H}$$
$$\boldsymbol{J}_i(\boldsymbol{v}(t),t) = \boldsymbol{H}^\dagger \boldsymbol{J}_{\tilde{i}}(\boldsymbol{H}\boldsymbol{v}(t),t)\boldsymbol{H}$$

and $\boldsymbol{J}_{\tilde{q}}$ and $\boldsymbol{J}_{\tilde{i}}$ are block diagonal.

Proof. The result follows directly from the relations between $\tilde{\boldsymbol{q}}$ and $\boldsymbol{q}$, and between $\tilde{\boldsymbol{i}}$ and $\boldsymbol{i}$. □

Remark. Computationally, Proposition 3.2.1 and its corollaries demonstrate how it is possible to compute the nodal sums of charge and current for the connected circuit by evaluating the constitutive relations for the unconnected circuit.

Example. A decomposition of the example circuit shown in Figure 3-1 is shown in Figure 3-4. Note that the subcircuits internal to the grid have one internal node and two terminal nodes, the subcircuits on the east and south border of the grid have one internal node and one terminal node, and the subcircuit on the south-east corner of the grid has one internal node and no terminal nodes.

Similar results to Corollary 3.2.2 and Corollary 3.2.3 can be constructed for other types of grid circuits as well (such as hexagonal grids).

3.3 Numerical Algorithms

The system of equations that describes an $N \times N$ grid circuit, constructed as in Corollary 3.2.2, can be written compactly as

$$\frac{d}{dt}\boldsymbol{q}\left(\boldsymbol{v}(t),t\right) + \boldsymbol{i}\left(\boldsymbol{v}(t),t\right) = 0. \tag{3.4}$$

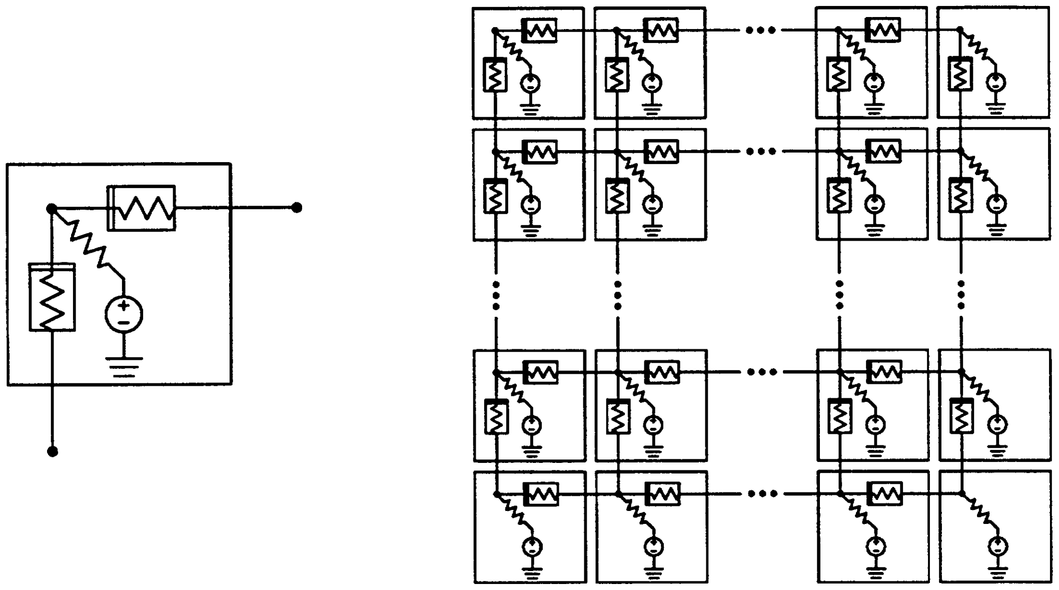

Figure 3-4: Decomposition of the example circuit shown in Figure 3-1. Note that the subcircuits on the east and south borders of the grid differ from the subcircuits elsewhere in the grid.

Here, n is the total number of nodes in the circuit, $\boldsymbol{v}(t), \boldsymbol{q}(\boldsymbol{v}(t), t), \boldsymbol{i}(\boldsymbol{v}(t), t) \in \mathbb{R}^n$ are the vectors of node voltages, sums of node charges, and sums of node resistive currents, respectively.

The transient simulation of the analog grid involves numerically solving (3.4). Discretizing (3.4) with the trapezoidal rule leads to the following algebraic problem for each time step h:

$$\frac{2}{h}\left[\boldsymbol{q}(\boldsymbol{v}(t+h), t+h) - \boldsymbol{q}(\boldsymbol{v}(t), t)\right] + \left[\boldsymbol{i}(\boldsymbol{v}(t+h), t+h) + \boldsymbol{i}(\boldsymbol{v}(t), t)\right] = 0. \qquad (3.5)$$

As is standard, the algebraic problem is solved with Newton's method,

$$\boldsymbol{J}_F(\boldsymbol{v}^m(t+h), t+h)\left[\boldsymbol{v}^{m+1}(t+h) - \boldsymbol{v}^m(t+h)\right] = -\boldsymbol{F}(\boldsymbol{v}^m(t+h), t+h) \qquad (3.6)$$

where

$$\begin{aligned} &\boldsymbol{F}(\boldsymbol{v}(t+h), t+h) = \qquad (3.7)\\ &\quad \frac{2}{h}\left[\boldsymbol{q}(\boldsymbol{v}(t+h), t+h) - \boldsymbol{q}(\boldsymbol{v}(t), t)\right] + \left[\boldsymbol{i}(\boldsymbol{v}(t+h), t+h) + \boldsymbol{i}(\boldsymbol{v}(t), t)\right] \end{aligned}$$

and the Jacobian $\boldsymbol{J}_F(\boldsymbol{v}(t+h), t+h)$ is

$$\boldsymbol{J}_F(\boldsymbol{v}(t+h), t+h) = \frac{2}{h}\frac{\partial \boldsymbol{q}(\boldsymbol{v}(t+h), t+h)}{\partial \boldsymbol{v}} + \frac{\partial \boldsymbol{i}(\boldsymbol{v}(t+h), t+h)}{\partial \boldsymbol{v}} \qquad (3.8)$$

In "classical" circuit simulators such as SPICE [5], the linear system of equations for each Newton iteration is solved by some form of sparse Gaussian elimination. Sparse

Algorithm 3.3.1 (Conjugate Gradient Squared algorithm for solving $\boldsymbol{Ax} = \boldsymbol{b}$).

```
$\boldsymbol{x}_0 = \boldsymbol{0}$
$\boldsymbol{r}_0 = \boldsymbol{b}$
$\boldsymbol{q}_0 = \boldsymbol{p}_{-1} = \boldsymbol{0}$
$\rho_{-1} = 0$
For $k = 0, 1, \ldots$
    if converged, break
    $\rho_k = \langle \tilde{\boldsymbol{r}}_0, \tilde{\boldsymbol{r}}_k \rangle$, $\beta_k = \rho_k / \rho_{k-1}$
    $\boldsymbol{u}_k = \boldsymbol{r}_k + \beta_k \boldsymbol{q}_k$
    $\boldsymbol{p}_k = \boldsymbol{u}_k + \beta_k(\boldsymbol{q}_k + \beta_k \boldsymbol{p}_{k-1})$
    $\boldsymbol{v}_k = \boldsymbol{A}\boldsymbol{p}_k$
    $\sigma_k = \langle \tilde{\boldsymbol{r}}_0, \boldsymbol{v}_k \rangle$, $\alpha_k = \rho_k / \sigma_k$
    $\boldsymbol{q}_{k+1} = \boldsymbol{u}_k - \alpha_k \boldsymbol{v}_k$
    $\boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_k \boldsymbol{A}(\boldsymbol{u}_k + \boldsymbol{q}_{k+1})$
    $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k(\boldsymbol{u}_k + \boldsymbol{q}_{k+1})$
```

Gaussian elimination is not well-suited to the present problem for several reasons. First, because of the structure of the matrix generated by grid-type circuits, sparse Gaussian elimination creates substantially more fill than for typical circuits with an equivalent number of nodes. Second, it is well known that sparse Gaussian elimination is difficult to parallelize. Therefore, a parallelizable iterative method is sought, such as a conjugate-direction method [6].

There are several conjugate-direction algorithms which are suitable for use as a linear system solver for grid circuits. Since, in the general case, the grid circuits may be constructed from non-reciprocal elements (e.g., MOS transistors), only methods suitable for non-symmetric matrices are considered, e.g., CGNR, GCR, GCR(k), ORTHOMIN, CGS [6, 7]. The present discussion will be restricted to CGS, presented in Algorithm 3.3.1, since experience has shown that it is the most efficient of the methods examined (see also [8, 9]).

One reason for the suitability of conjugate direction methods to the problem under consideration is that the convergence rates of these methods depend on the condition number of $\boldsymbol{J}_F$. By taking timesteps small enough, the capacitive portion of the Jacobian, which tends to be better conditioned than the conductive portion, will dominate (see [10]).

To demonstrate the effectiveness of the conjugate-direction methods, in Table 3-1 the CPU time required to compute the transient analysis of the network in Figure 3-1 is compared using several different matrix solution algorithms to solve (3.6). The conjugate-

Size	Direct	CG	ILUCG	CGS	ILUCGS
16×16	2.69	0.94	0.85	1.12	1.07
32×32	39.3	4.10	3.93	5.08	4.69
64×64	464	24.5	24.2	27.8	30.6
128×128	25000	713	1340	1420	1790

Table 3-1: Comparison of serial execution time for the transient simulation of the circuit in Figure 3-1, when using direct, CG, ILUCG, CGS, and ILUCGS linear system solvers. For this example, $g_f = 3.0 \times 10^{-5}\Omega^{-1}$ and g_s has a conductance of $1.0 \times 10^{-3}\Omega^{-1}$ when linearized about zero. All execution times are CPU seconds for an IBM RS/6000 model 530 workstation.

direction methods shown in Table 3-1 are conjugate gradient (CG), incomplete LU preconditioned CG (ILUCG), conjugate gradient squared (CGS), and incomplete LU preconditioned CGS (ILUCGS). As the table indicates, sparse Gaussian elimination is much slower than the conjugate-direction methods, especially for larger problem sizes.

3.4 Implementation

For an algorithm to approach peak parallel performance on a SIMD machine, it must satisfy three requirements. First, the problem must have enough parallelism to effectively use the available processors. Second, the algorithm should depend only on local or infrequent interprocessor communication. And third, the algorithm must be mostly *data-parallel*, meaning:

- One can identically map individual pieces of data to individual processors for all relevant processors, and

- One can operate identically on the data with all the relevant processors

The general circuit simulation problem violates all three of the above constraints, and previous attempts at using a SIMD machine to accelerate circuit simulation have not yielded impressive results [10, 11]. As will be shown in the rest of this section, simulation of grid-based analog signal processors *is* well suited to SIMD architectures. These circuits are large enough to keep a large number of processors active, and they can be simulated with algorithms that depend on nearest-neighbor communication between processors.

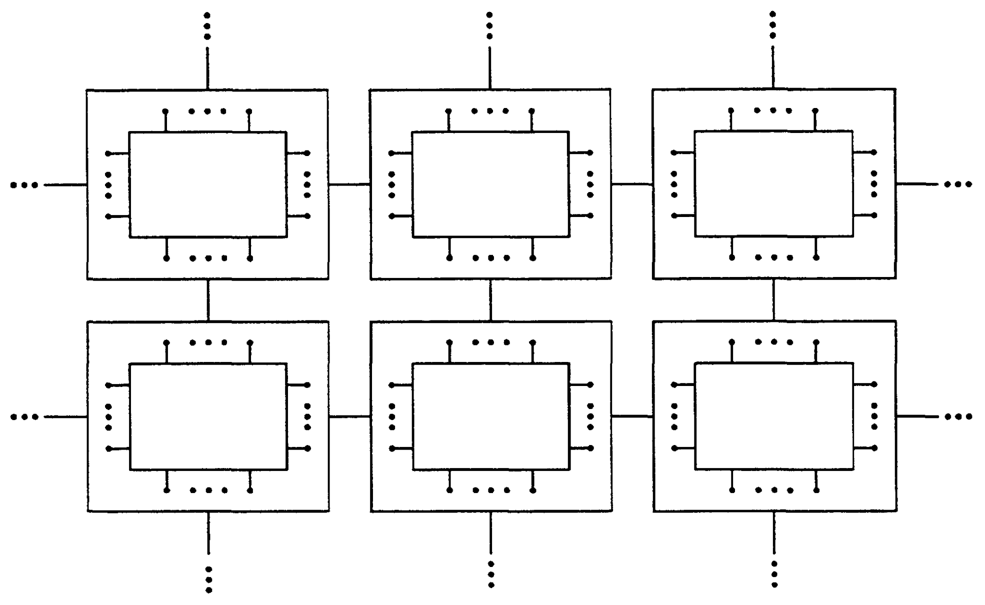

Figure 3-5: Mapping of subcircuits to processor grid. Each processor, represented by a single box, contains the data necessary for simulating a single subcircuit. The lines connecting the boxes represent the inter-processor communication network.

3.4.1 Data to Processor Mapping

The two-dimensional nature of grid-based analog signal processing circuits maps naturally onto a two-dimensional grid-connected processor network so that data parallelism and locality are maintained. In particular, assume that it is desired to simulate an $N \times N$ grid circuit that is constructed by replicating identical subcircuits as described in Section 3.2. The problem is mapped onto an $N \times N$ processor array by assigning the data for one subcircuit to each processor (see Figure 3-5).

The characteristics of the grid circuit can be obtained with this mapping by using Corollary 3.2.2, as described in Sections 3.4.2 and 3.4.3. Note that $\tilde{\boldsymbol{v}}$, $\tilde{\boldsymbol{q}}(\tilde{\boldsymbol{v}})$, and $\tilde{\boldsymbol{i}}(\tilde{\boldsymbol{v}})$ can be completely represented by keeping $(M_{int} + M_{term})$ components of these vectors on each processor. However, representing $\boldsymbol{v}$, $\boldsymbol{q}(\boldsymbol{v})$, and $\boldsymbol{i}(\boldsymbol{v})$ requires only M_{int} components of each vector on each processor. Therefore, the convention is adopted that the components of $\tilde{\boldsymbol{v}}$, $\tilde{\boldsymbol{q}}(\tilde{\boldsymbol{v}})$, and $\tilde{\boldsymbol{i}}(\tilde{\boldsymbol{v}})$ which correspond to the internal nodes are stored in locations $\{1, \ldots, M_{int}\}$ and the components which correspond to terminal nodes are stored in locations $\{M_{int} + 1, \ldots, M_{int} + M_{term}\}$. With this convention, the vectors $\boldsymbol{v}$, $\boldsymbol{q}(\boldsymbol{v})$, and $\boldsymbol{i}(\boldsymbol{v})$ are simply the first n locations of the vectors $\tilde{\boldsymbol{v}}$, $\tilde{\boldsymbol{q}}(\tilde{\boldsymbol{v}})$, and $\tilde{\boldsymbol{i}}(\tilde{\boldsymbol{v}})$, respectively.

3.4.2 Device Evaluation

The device evaluation stage of circuit simulation involves the computation of the right-hand side and the Jacobian for the Newton iteration (3.7), i.e., computing $\boldsymbol{F}$ and $\boldsymbol{J_F}$ as in (3.7) and (3.8). Since the linear system solver is a conjugate-direction iterative method, $\boldsymbol{J_F}$ is not needed explicitly, rather, it is only necessary to calculate the matrix-vector product $\boldsymbol{y} = \boldsymbol{J_F x}$. If (by definition) $\boldsymbol{J}_{\tilde{F}} = \frac{2}{h}\boldsymbol{J}_{\tilde{q}} + \boldsymbol{J}_{\tilde{\imath}}$, then by Corollary 3.2.3, the matrix-vector product $\boldsymbol{y} = \boldsymbol{J_F x}$ can be calculated according to $\boldsymbol{y} = \boldsymbol{H}^T \boldsymbol{J}_{\tilde{F}} \boldsymbol{H x}$. Therefore, only $\boldsymbol{J}_{\tilde{F}}$ is calculated during the device evaluation process.

Using the topological matrix described in Corollary 3.2.3, the computation and communication required to compute $\boldsymbol{F}$ and $\boldsymbol{J}_{\tilde{F}}$ can be described by the following, where the dependence on t is omitted for clarity:

1. Calculate $\tilde{\boldsymbol{v}} = \boldsymbol{H v}$. In this step, the values of $\tilde{\boldsymbol{v}}$ corresponding to terminal nodes are set with nearest-neighbor communication operations.

2. Calculate the Jacobian, $\boldsymbol{J}_{\tilde{F}}(\tilde{\boldsymbol{v}}) = \frac{2}{h}\boldsymbol{J}_{\tilde{q}}(\tilde{\boldsymbol{v}}) + \boldsymbol{J}_{\tilde{\imath}}(\tilde{\boldsymbol{v}})$ and "right-hand-side" components, $\tilde{\boldsymbol{q}}(\tilde{\boldsymbol{v}})$ and $\tilde{\boldsymbol{\imath}}(\tilde{\boldsymbol{v}})$, by evaluating the cell devices in parallel.

3. Calculate $\boldsymbol{q}(\boldsymbol{v}) = \boldsymbol{H}^T \tilde{\boldsymbol{q}}(\tilde{\boldsymbol{v}})$ and $\boldsymbol{i}(\boldsymbol{v}) = \boldsymbol{H}^T \tilde{\boldsymbol{\imath}}(\tilde{\boldsymbol{v}})$. In this step, the values of $\tilde{\boldsymbol{q}}(\tilde{\boldsymbol{v}})$ and $\tilde{\boldsymbol{\imath}}(\tilde{\boldsymbol{v}})$ corresponding to terminal nodes are communicated with nearest-neighbor communication operations and added into the appropriate locations of $\boldsymbol{q}(\boldsymbol{v})$ and $\boldsymbol{i}(\boldsymbol{v})$.

An explicit representation of $\boldsymbol{H}$ is not needed to accomplish the communication operations — a local representation of how nodes are connected to each other across the processor boundary is the only requirement.

3.4.3 Linear System Solution

There are two parts of the CGS iteration which involve parallel data: the vector inner product and the matrix-vector product. The vector inner-product is computed with an in-place multiply and a global sum. The matrix-vector product $\boldsymbol{y} = \boldsymbol{J_F x}$ is computed according to $\boldsymbol{y} = \boldsymbol{H}^T \boldsymbol{J}_{\tilde{F}} \boldsymbol{H x}$, using the result of Corollary 3.2.3, with the following sequence of operations:

1. Calculate $\tilde{\boldsymbol{x}} = \boldsymbol{H x}$.

2. Perform parallel block matrix-vector multiplication, $\tilde{\boldsymbol{y}} = \boldsymbol{J}_{\tilde{F}} \tilde{\boldsymbol{x}}$.

3. Calculate $\boldsymbol{y} = \boldsymbol{H}^T \tilde{\boldsymbol{y}}$.

Here, $\boldsymbol{x}$, $\tilde{\boldsymbol{x}}$ $\boldsymbol{y}$ and $\tilde{\boldsymbol{y}}$ are stored and $\boldsymbol{Hx}$ and $\boldsymbol{H}^T\tilde{\boldsymbol{y}}$ are calculated just as in the device evaluation process above. Again, steps 1 and 3 involve explicit communication operations, but 2 does not involve any communication, since $\boldsymbol{J}_{\tilde{F}}$ is block diagonal.

Block Diagonal Preconditioning

An effective technique for improving the performance of conjugate-direction iterative methods is the use of preconditioners. That is, instead of solving $\boldsymbol{Ax} = \boldsymbol{b}$ directly, the conjugate-direction method is applied to the equivalent system

$$\boldsymbol{P}^{-1}\boldsymbol{Ax} = \boldsymbol{P}^{-1}\boldsymbol{b}.$$

Typically, $\boldsymbol{P}$ is chosen so that the system $\boldsymbol{Pz} = \boldsymbol{r}$ for $\boldsymbol{z}$ is easy to solve and so that the conjugate direction method applied to the preconditioned system converges faster than when applied to the non-preconditioned system.

The general structure of the Jacobian matrix $\boldsymbol{J}_F$ is that of a block diagonal matrix with block off-diagonal bands. The diagonal blocks are of size $M_{int} \times M_{int}$; the block off-diagonal bands are of size $M_N \times M_N$, $M_E \times M_E$, $M_W \times M_W$, and $M_S \times M_S$. This suggests a block iterative method for solving (3.6), using the diagonal blocks of $\boldsymbol{J}_F$ as the preconditioner. A block iterative scheme is particularly well suited to a SIMD implementation, since each block can be solved simultaneously in parallel. Note that although $\boldsymbol{J}_{\tilde{F}}$ is already block diagonal, inverting its blocks is not the same as inverting the block diagonal portion of $\boldsymbol{J}_F$.

Rather, let $\boldsymbol{J}_F = \boldsymbol{P} + \boldsymbol{R}$, where $\boldsymbol{P}$ is the block diagonal part of $\boldsymbol{J}_F$. To use $\boldsymbol{P}$ as a preconditioner in the CGS algorithm, it must be formed from $\boldsymbol{J}_{\tilde{F}}$. Let $\tilde{R}_{j,k}$ be the part of $\boldsymbol{J}_{\tilde{F}}$ corresponding to the coupling between internal and terminal nodes of the same subcircuit — this coupling will become the off-diagonal coupling blocks in $\boldsymbol{J}_F$. Define $\tilde{\boldsymbol{P}}$ by $\boldsymbol{J}_{\tilde{F}} = \tilde{\boldsymbol{P}} + \tilde{\boldsymbol{R}}$ (see Figure 3-6). Using Corollary 3.2.3, note that

$$\boldsymbol{P} = \boldsymbol{H}^T\tilde{\boldsymbol{P}}\boldsymbol{H}. \tag{3.9}$$

Solving the linear system $\boldsymbol{Pz} = \boldsymbol{r}$ for $\boldsymbol{z}$ is then accomplished by:

1. Form $\boldsymbol{P} = \boldsymbol{H}^T\tilde{\boldsymbol{P}}\boldsymbol{H}$.

2. Solve $\boldsymbol{Pz} = \boldsymbol{r}$.

Comments on Preconditioning

On serial machines, conjugate-direction methods have benefitted greatly from the use of preconditioning. However, the author's experience, as well as the results reported in [12],

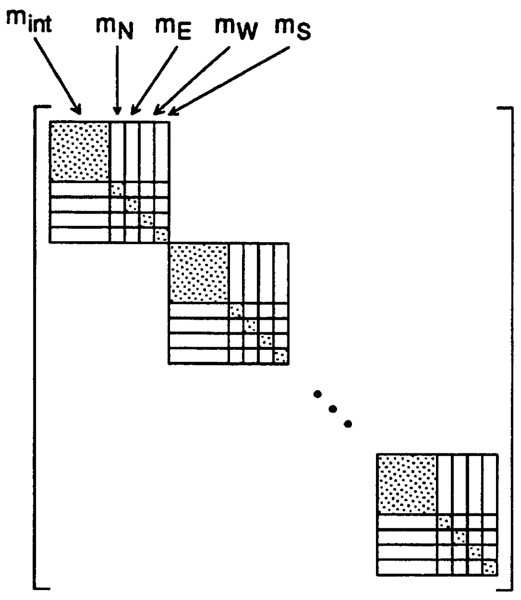

Figure 3-6: Definition of $\tilde{\boldsymbol{P}}$. Since the coupling between internal nodes and terminal nodes will become off-diagonal blocks in $\boldsymbol{J}_F$, these coupling elements are removed from $\boldsymbol{J}_{\tilde{F}}$ to form $\tilde{\boldsymbol{P}}$.

indicate that some the advantages of using a preconditioner are lost when implementing conjugate-direction methods on a massively parallel machine. In this section, results from [13] regarding parallel algorithms for solving PDE's are applied to provide some explanations about the limitations of a parallel preconditioner. The main idea is that there are two bounds on algorithmic performance — communication bounds and computation bounds — and preconditioning a conjugate-direction method cannot improve the communication lower bound.

As a model problem, consider the Dirichlet problem in two dimensions:

$$\begin{aligned} \nabla^2 u &= f \quad (x,y) \in \Omega \subset \mathbb{R}^2 \\ u|_{\partial\Omega} &= 0. \end{aligned} \tag{3.10}$$

For simplicity, let $\Omega = I^2 \subset \mathbb{R}^2$. To solve (3.10), apply a finite-difference scheme with stepsize h to obtain:

$$\boldsymbol{A}\boldsymbol{u} = \boldsymbol{f} \tag{3.11}$$

where $n = h^{-2}$, $\boldsymbol{A} \in \mathbb{R}^{n \times n}$, and $\boldsymbol{u}, \boldsymbol{f} \in \mathbb{R}^n$. The linear system of equations in (3.11) can also be realized with a grid of linear resistors.

Consider the behavior of the CG algorithm for solving (3.11). As is well known, the number of iterations required to solve (3.11) for a fixed convergence criterion, say ϵ, is $\mathcal{O}\left(\sqrt{\kappa(\boldsymbol{A})}\right)$, where $(\kappa(\boldsymbol{A}))$ is the condition number of $\boldsymbol{A}$ [14]. For the discretized

Laplacian, $\kappa(A) = ch^{-2}$, so that the number of iterations required to solve (3.11) for a fixed ϵ is $\mathcal{O}(\sqrt{n})$.

The premier preconditioning techniques for CG applied to the discretized Laplacian are the incomplete factorization methods, such as modified incomplete Cholesky (MICCG). What makes these preconditioners so powerful is that they lower the complexity of the condition number of the system to be solved. In fact, the serial implementation of the MICCG method is known to solve (3.11) for fixed ϵ in $\mathcal{O}\left(n^{\frac{1}{4}}\right)$ iterations.

Unfortunately, this complexity improvement is not necessarily achievable with a parallel preconditioned CG algorithm. If sufficient processors are available such that each iteration of CG takes constant time, then the computational complexity for CG on a parallel machine is the number of iterations required, i.e., $\mathcal{O}(\sqrt{n})$, and the communication lower bound is the maximum network distance that information must travel so that each processor can compute its solution. For the discretized Laplacian, this distance is the maximum network distance between any two processors, since each processor must at some point get information from every other processor to compute the solution [13]. This implies that the communication lower bound for CG is $\Omega(\sqrt{n})$ on a two-dimensional network or $\Omega(\log n)$ on a hypercube network. Now, for preconditioned CG, the computational complexity may be less than the $\mathcal{O}(\sqrt{n})$ for plain CG (e.g., $\mathcal{O}(n^{\frac{1}{4}})$ for MICCG). However, the communication lower bound is the same, i.e., $\Omega(\sqrt{n})$.

The conclusion is that a preconditioner which only uses a two-dimensional network communication pattern, as is the case with incomplete factorization methods, cannot reduce the complexity of the parallel CG algorithm applied to the discretized Laplacian. However, a preconditioner which uses a hypercube network communication pattern, such as a hierarchical basis preconditioner, can at least improve the communication lower bound. These conclusions are borne out in the experiments reported in [12]. Whether those preconditioners which improve the communication lower bound will also provably lower the computational complexity is an open question.

3.4.4 Grid Boundaries

To this point, what happens on the boundaries of the grid has been glossed over. Since the subcircuits represent physical circuitry, the boundary subcircuits will differ from the subcircuits internal to the grid (as in the grid of nonlinear resistors shown in Figure 3-4).

To handle the boundary subcircuits properly, the grid subcircuits are subdivided into smaller sub-subcircuits, some of which can be omitted from the boundary subcircuits in order to produce correct circuit behavior. Since all processors contain identical data, the boundary processors do, in effect, still have all the sub-subcircuits. However, by turning off the appropriate boundary processors whenever data corresponding to an omitted sub-subcircuit is manipulated, the correct circuit behavior at the grid boundaries can be produced. For reasons of clarity, further discussion of this operation is omitted; it is performed in a straightforward manner for all the algorithms to be presented.

3.5 Experimental Results

In this section, experimental results are presented that compare serial and parallel execution times for several types of grid-based analog signal processors. To make the results as meaningful as possible, the fastest serial algorithm is compared with the fastest parallel algorithm. The programs used to test the serial and parallel programs both use MIT's SIMLAB program [15, 16] as a base — in other words, to a large extent the serial and parallel programs are the *same* program. This is done to guarantee that the code is as similar as possible for the two programs.

The parallel algorithms were executed on the Connection Machine model CM-2 — a single-instruction multiple data (SIMD) parallel computer which, in its largest configuration, contains 65,536 bit-serial processors and 2048 Weitek floating-point units (FPU's) [17]. The bit-serial processors are clustered together into groups of 16 within a single integrated circuit, and these IC's are connected together in a 12-dimensional hypercube. Two IC's, or 32 processors, share a single Weitek FPU. Note that a fully configured CM-2 contains 2048 times as much floating point hardware as a conventional computer containing a single Weitek FPU (e.g., a SUN-4).

The Connection Machine allows the user to map problems which are larger than the actual number of physical processors through a software emulation process which creates virtual processors. All CM experiments reported here used a virtual processor ratio of one, i.e., the problems were the same size as or smaller than the actual number of physical processors on the CM.

The serial experiments were run using the VSIM (Vision SIMulation) program. The VSIM program is essentially SIMLAB with idealized nonlinear elements and image I/O support added. The parallel experiments were run using the CMVSIM (Connection Machine Vision SIMulation) program [18, 19]. Implementation of the CMVSIM program required major enhancement to SIMLAB to support the parallel algorithms. The parallel portions of the code were written in C* Version 6.0, a CM superset of C [20, 21]. In those cases where the parallel code was to be a parallel version of code already existing in serial form in SIMLAB, care was taken to make it as much like the serial code as possible.

The serial experiments were run on a SUN-4/490 workstation, and the CM results were obtained on a 16K CM-2 with double-precision floating point hardware, using a SUN-4/490 as a front-end. The serial execution times were run on the same machine which was used as the front-end for the parallel experiments. All computations were performed in double precision arithmetic.

Example Circuit File

CMVSIM constructs the vision circuit grid by reading a description of the simple cell, which is partitioned by the user into separate circuit files so that the borders can be handled properly. Each of the simple cell files contains a complete circuit given in SIMLAB circuit syntax. The files are denoted by their relationship to the border of the cell, i.e., `north`, `east`, `west`, or `south`; the circuitry internal to the cell which does not vary on the borders of the grid is known as the `here` circuitry. The files containing the circuitry are given the extensions `.rel.n`, `.rel.e`, `.rel.w`, `.rel.s`, and `.rel.h` (with the obvious associated directions).

The `north`, `east`, `west`, and `south` circuit files contain the sub-circuits which connect cells together. In order to connect cells, the border circuits contain special elements called connector elements which specify how the terminal elements are connected across processor boundaries (see Figure 3-7).

Since the simple cell is divided into separate circuit files, CMVSIM must be told which nodes are common to all files. This is done with a "common" comment in the `here` circuit file. The "common" comment has the form:

; **common** ⟨*node*1⟩ [⟨*node*2⟩ ...]

Normally, the common nodes correspond to those nodes to which terminals from neighboring subcircuits are joined. Refer to the "CMVSIM users' guide" [19] for more details on using CMVSIM.

The contents of the circuit files to specify a resistive grid are shown in the files `lres.rel.e`, `lres.rel.s`, and `lres.rel.h`.

Linear Resistive Grid

Table 3-2 shows the results obtained while simulating the linear resistive circuit grid shown in Figure 3-8. The circuit for which the results are shown had a $1K$ resistance between neighboring nodes, a $33.33K$ resistance to ground, and a parasitic capacitance

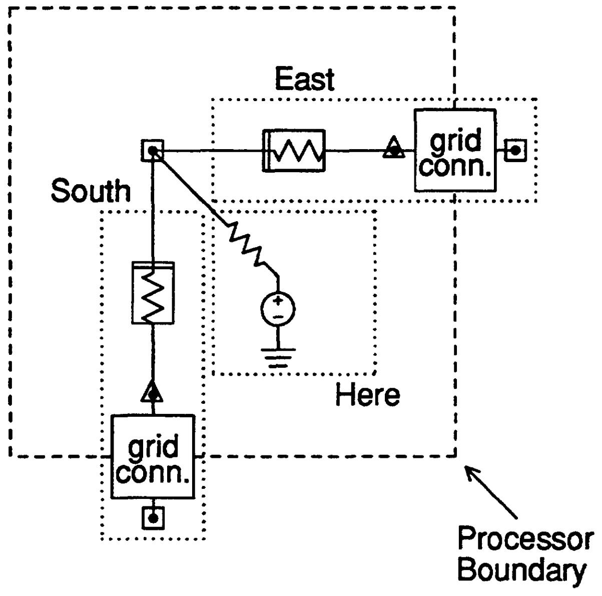

Figure 3-7: Example separation of a subcircuit into sub-subcircuits. This subcircuit is divided into three separate SIMLAB circuits: `east`, `south`, and `here`. The `east` and `south` circuits contain grid connectors which specify how the terminal elements are connected across processor boundaries.

of $1\times10^{-15}F$ at each node. A random image was introduced to the network and the startup transient of the first 1×10^{-5} seconds was simulated.

For the 128×128 grid — which is the largest that will fit directly on the 1/4 size CM-2 — the CM achieved a speedup of about a factor of 50. If the serial and parallel results are extrapolated to a 256×256 grid — which is the largest that will fit directly on a full size CM-2 — the CM should be able to achieve a speedup of over a factor of 200.

	Serial			CM
Size	Direct	CG	ILCG	CG
16×16	9.95	5.6	4.55	10.30
32×32	107.15	30.0	25.57	10.46
64×64	1322.28	126.25	113.22	9.26
128×128	(1.63×10^4)	(531)	(501)	9.54
256×256	(2.01×10^5)	(2236)	(2220)	(10)

Table 3-2: Experimental result: Linear circuit grid.

lres.rel.e

```
global 0

r1 here east r r=1k
c0 east here connector
```

lres.rel.s

```
global 0

r1 here south r r=1k
c0 south here connector
```

lres.rel.h

```
global 0

; common here

r1 here 1 r r=1k
v1 1 0 dc v=2

cmvsim options cmin = 1.e-6
plot here
```

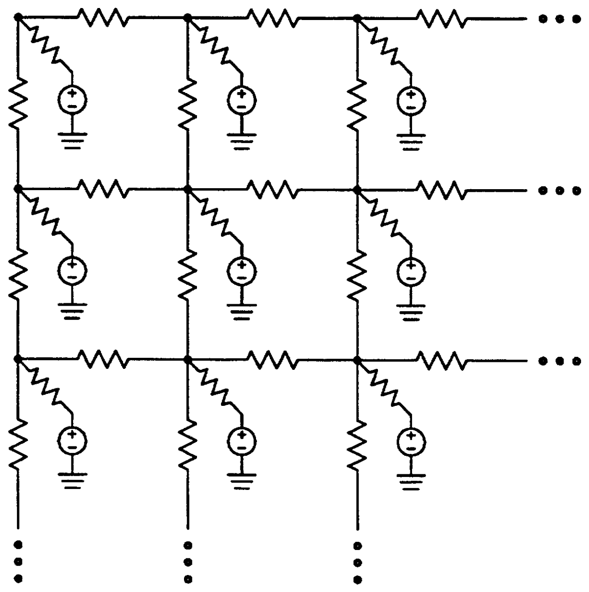

Figure 3-8: Linear resistive grid.

	Serial			CM
Size	Direct	CG	ILCG	CG
16×16	183.13	176.03	126.15	227.88
32×32	4027.23	1802.10	1445.75	487.16
64×64	(8.86×10^4)	14287.90	10377.90	896.01
128×128	(1.95×10^6)	(1.13×10^5)	(7.45×10^4)	1445.12
256×256	(4.28×10^7)	(8.99×10^5)	(5.35×10^5)	(2330)

Table 3-3: Experimental result: Nonlinear circuit grid.

Nonlinear Resistive Grid

Table 3-3 shows the results obtained while simulating the nonlinear resistive circuit grid shown in Figure 3-1. The nonlinear resistance has the following characteristic equation:

$$i(v) = \frac{\alpha v}{1 + e^{-\beta(\gamma - \alpha v^2)}}. \tag{3.12}$$

The circuit for which the results are shown had a resistance of $10k$ to ground, a parasitic capacitance of 1×10^{-7} and parameter values for the nonlinear resistance of $\alpha = 1\times10^{-3}$, $\gamma = 2\times10^{-5}$ with β being swept from 0 to 1×10^5 over the simulation interval. A random image was introduced to the network, the DC solution was found with $\beta = 0$ (i.e., so that the network was linear), and then a transient simulation of one second was performed, during which β was increased from 0 to 1×10^5. The reasons for this type of simulation are explained in detail in [22].

For the 128×128 grid, the CM achieved a speedup of about a factor of 50. If the serial and parallel results are extrapolated to a 256×256 grid, the CM should be able to achieve a speedup of over a factor of 200.

Mead's Silicon Retina

Tables 3-4 and 3-5 show the results obtained while simulating Mead's Silicon Retina as described in [1] and shown in Figure 3-9. Table 3-4 shows results obtained when presenting the circuit with a constant image while Table 3-5 shows the results obtained when presenting the circuit with a random input image. The simulations used the same SPICE MOS level 3 devices in both the serial and parallel versions.

The Silicon Retina example contains a significant amount of circuitry in each cell and the block diagonal preconditioner is quite effective in reducing the CPU time required to compute the solution. For the examples with the constant input image, the block diagonal preconditioner reduced the CPU time by approximately a factor of two over

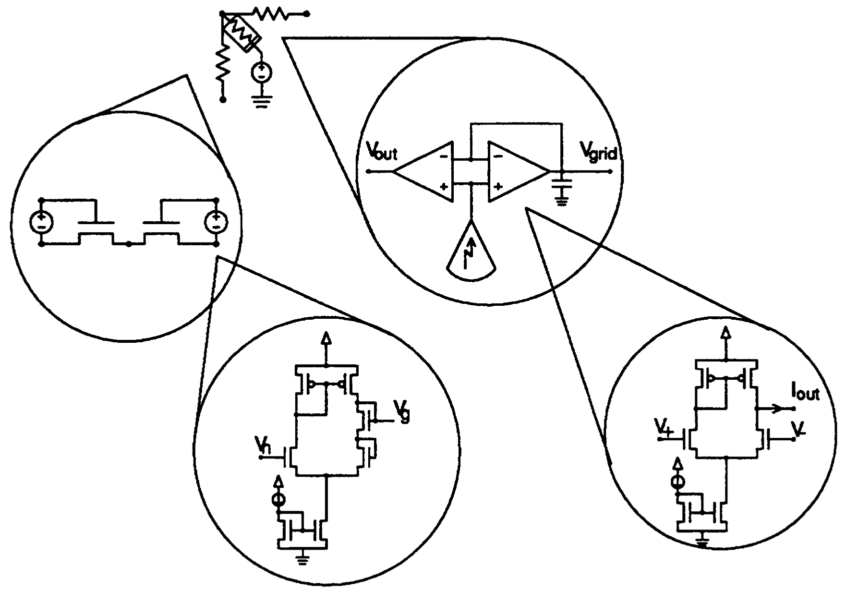

Figure 3-9: Mead's Silicon Retina. The simulated chip contains 23 MOS level 3 devices per cell for a total of 376,320 devices and 261,888 nodes in a 128×128 grid.

	Serial			CM	
Size	Direct	CGS	ILCGS	CGS	PCGS
16×16	516.15	1911.72	605.22	840.34	436.225
32×32	3353.68	8241.03	2532.37	936.89	454.67
64×64	(21790)	(35525)	(10596)	1020.88	461.42
128×128	(141583)	(153142)	(44336)	1048.25	463.89
256×256	(919959)	(660172)	(185510)	(1076)	(466)

Table 3-4: Experimental result: Mead's Silicon Retina with constant input image.

plain CGS for all grid sizes. For the examples with the random input image, the block diagonal preconditioner reduced the CPU time by up to a factor of four over plain CGS.

For the 128 × 128 grid, the CM achieved a speedup of about a factor of 95 for the circuit with the constant input image and a speedup of over a factor of 144 for the circuit with the random input image. If the serial and parallel results are extrapolated to a 256 × 256 grid, the CM should be able to achieve a speedup of about a factor of 400 for the circuit with the constant input image and a speedup of over a factor of 650 for the circuit with the random input image. If one assumes that the Sun4/490 workstation on which the serial results were obtained can attain about 2 MFlops floating point performance then the factor of 650 speedup represents over a gigaflop for the full-sized CM-2.

	Serial			CM	
Size	Direct	CGS	ILCGS	CGS	PCGS
16×16	1239.50	4405.77	1244.75	1845.34	894.42
32×32	10713.1	25740.9	7343.03	2414.55	1221.14
64×64	(9.2×10^4)	(1.5×10^5)	(4.3×10^4)	4787.49	1303.49
128×128	(8.0×10^5)	(8.8×10^5)	(2.5×10^5)	6661.22	1730.36
256×256	(6.9×10^6)	(5.1×10^6)	(1.5×10^6)	(9268)	(2297)

Table 3-5: Experimental result: Mead's Silicon Retina with random input image.

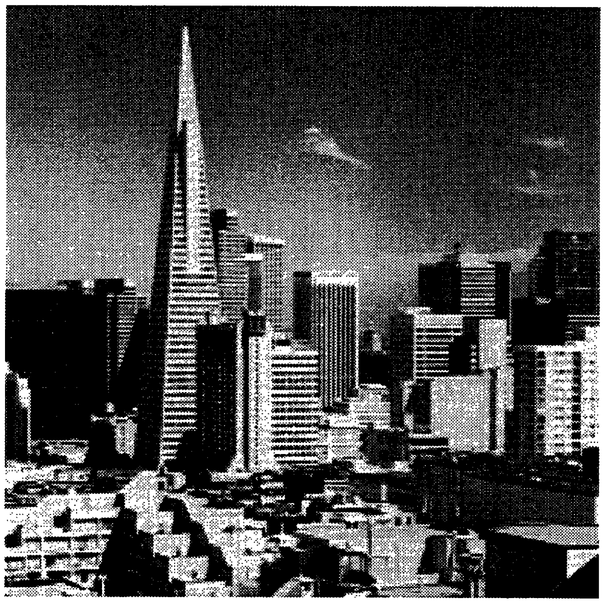

Figure 3-10: 256×256 image of the San Francisco sky line.

Processing Images

One of the most significant features of CMVSIM is that the program can be used to investigate how particular vision circuits process images. To demonstrate this use of CMVSIM, the results of two experiments are presented where the same network is subjected to two different types of continuations, resulting in drastically different output images (see [22] for detailed treatment of the continuations). The network used is shown in Figure 3-1 and uses the nonlinear element described by (3.12); let the linear conductance to ground be called λ_f. Figure 3-10 shows a 256×256 image — a portion of the San Francisco sky line — used as the input image.

Figure 3-11 shows the output produced by the idealized nonlinear network with a continuation performed on the value of β. The fixed parameter values were $\alpha = 1\times10^{-3}$, $\gamma = 1\times10^{-5}$, $\lambda_f = 3\times10^{-5}$; the value of β was linearly varied as a function of time from $\beta = 0$ to $\beta = 1\times10^{6}$. Figure 3-11(a) shows the output of the network at the beginning of the continuation when $\beta = 0$; Figure 3-11(b) shows the output of the network at an intermediate point of the continuation when $\beta = 2\times10^{4}$; Figure 3-11(c) shows the output of the network at the end of the continuation when $\beta = 1\times10^{6}$.

Figure 3-12 shows the output produced by the idealized nonlinear network with a continuation performed on the value of λ_f. The fixed parameter values were $\alpha = 1\times10^{-3}$, $\gamma = 1\times10^{-5}$, and $\beta = 1\times10^{6}$; the value of the λ_f was linearly varied as a function of time from $\lambda_f = 1$ to $\lambda_f = 3\times10^{-5}$. Figure 3-12(a) shows the output of the network at the beginning of the continuation when $\lambda_f = 1$; Figure 3-12(b) shows the output of the network at an intermediate point of the continuation when $\lambda_f = 1\times10^{-3}$; Figure 3-12(c) shows the output of the network at the end of the continuation when $\lambda_f = 3\times10^{-5}$. Note that the final parameter values of this network are identical to those for the network of Figure 3-11.

Discussion

The 128×128 example of Mead's Silicon Retina contains well over 300,000 MOS level 3 devices, but the examples with constant and random input images were simulated in eight minutes and 29 minutes, respectively.

It is interesting to note that the improvement provided by CMVSIM over VSIM was quite a bit higher for the Silicon Retina examples than for the linear and nonlinear resistive grid examples. One reason for this is that the Silicon Retina examples provide a much higher computation to communication ratio than the linear and nonlinear resistive grids. The linear and nonlinear grids each have three simple devices, one internal node, and two connecting nodes per subcircuit, whereas the Silicon Retina has 23 MOS level 3 devices, 16 internal nodes, and four connecting nodes per subcircuit. Moreover, simulation of the Silicon Retina with the block diagonal preconditioner also involves a parallel linear system solution step. Maintaining a large computation to communication ratio is especially important when using the Connection Machine since the CM has notoriously slow communication functions.

At present, CMVSIM only supports Manhattan-style circuit grids. However, enhancements to allow other regular grid geometries, such as hexagonal grids, have been anticipated and should be able to be incorporated in a straightforward manner. CMVSIM could also be enhanced to allow the simulation of circuits which are "mostly regular," such as

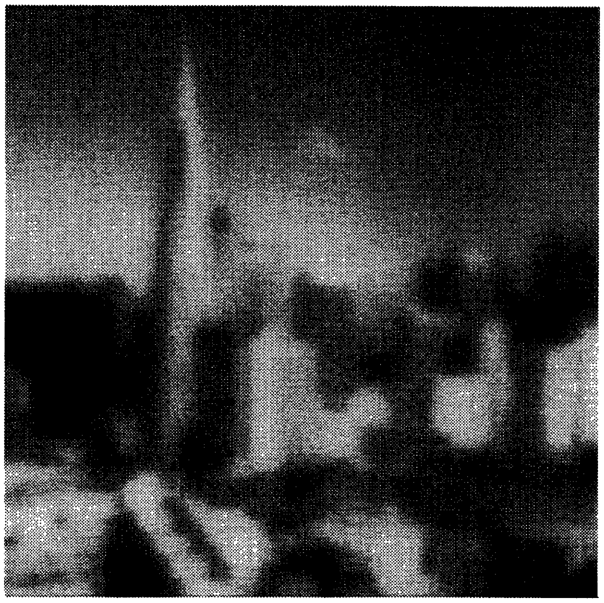

Figure 3-11(a): Output image produced by β-continuation network. Here, the parameter values are $\alpha = 1\times10^{-3}$, $\gamma = 1\times10^{-5}$, $\lambda_f = 3\times10^{-5}$, and $\beta = 0$.

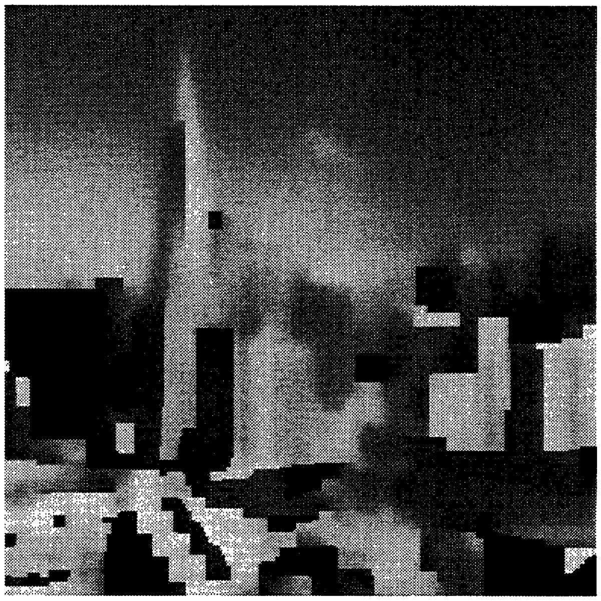

Figure 3-11(b): Output image produced by β-continuation network. Here, the parameter values are $\alpha = 1\times10^{-3}$, $\gamma = 1\times10^{-5}$, $\lambda_f = 3\times10^{-5}$, and $\beta = 2\times10^{4}$.

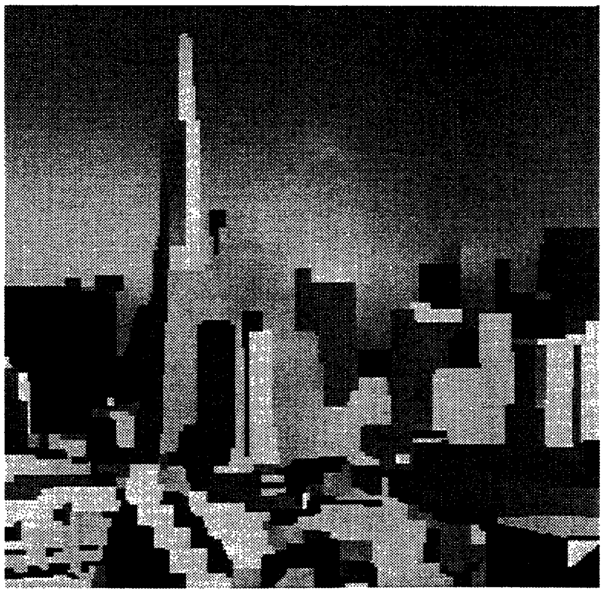

Figure 3-11(c): Output image produced by β-continuation network. Here, the parameter values are $\alpha = 1\times10^{-3}$, $\gamma = 1\times10^{-5}$, $\lambda_f = 3\times10^{-5}$, and $\beta = 1\times10^{6}$.

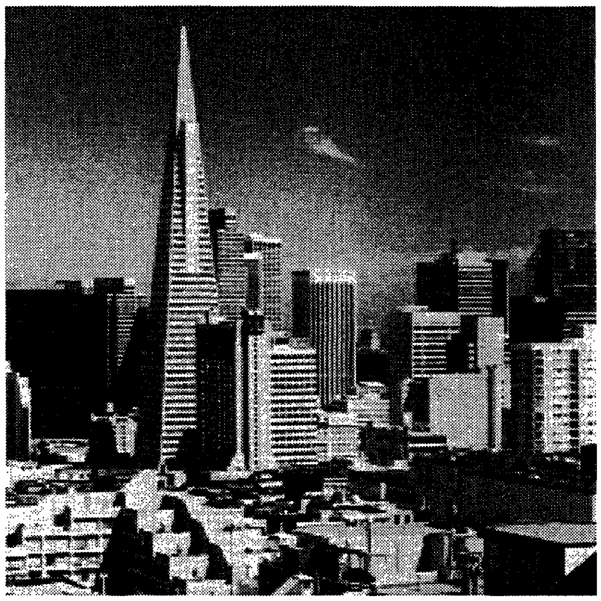

Figure 3-12(a): Output image produced by λ_f-continuation network. Here, the parameter values are $\alpha = 1 \times 10^{-3}$, $\gamma = 1 \times 10^{-5}$, $\beta = 1 \times 10^{6}$, and $\lambda_f = 1$.

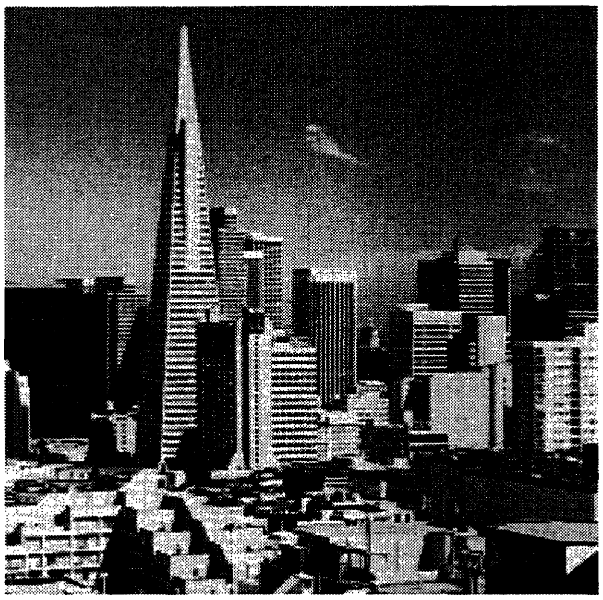

Figure 3-12(b): Output image produced by λ_f-continuation network. Here, the parameter values are $\alpha = 1 \times 10^{-3}$, $\gamma = 1 \times 10^{-5}$, $\beta = 1 \times 10^{6}$, and $\lambda_f = 1 \times 10^{-3}$.

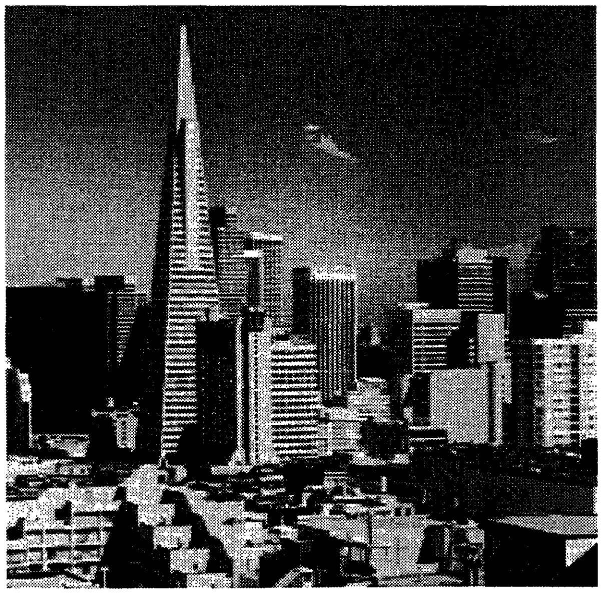

Figure 3-12(c): Output image produced by λ_f-continuation network. Here, the parameter values are $\alpha = 1 \times 10^{-3}$, $\gamma = 1 \times 10^{-5}$, $\beta = 1 \times 10^{6}$, and $\lambda_f = 3 \times 10^{-5}$. Note that the final parameter values of this network are identical to those for the network of Figure 3-11, but that the output image is much closer to the input image shown in Figure 3-10.

memory chips. To do this, CMVSIM would operate in a hybrid mode in which the regular portions of the circuit would be simulated on the CM and the non-regular portions would be simulated on the front-end.

3.6 Conclusion

Although the irregular structure of the general circuit simulation problem seems to preclude the use of massively parallel machines, such as the Connection Machine, for circuit simulation, there are nevertheless certain interesting (and increasingly important) classes of circuits which *do* benefit from these architectures. In this chapter, one such class of circuits, namely robotic vision chips, was examined.

In developing algorithms for these circuits, the fact that the circuits have a regular structure which maps nicely to a massively parallel architecture was exploited. Moreover, the coupling between cells in these arrays is such that block iterative methods could be used to solve the equations generated by an implicit time-discretization scheme.

The experimental results were very encouraging. It was possible to simulate a 128×128 example of Mead's Silicon Retina, a circuit having over 300,000 SPICE level 3 MOS devices, in about half an hour. The same circuit simulated with the best serial algorithm running on a Sun4/490 would take an estimated *three days* — assuming the workstation had enough memory to accommodate the problem at all. The simulation capability provided by CMVSIM should allow simulation to be an important part of the design cycle of real robotic vision chips for the first time.

References

[1] C. Mead, *Analog VLSI and Neural Systems.* Reading, MA: Addison-Wesley, 1988.

[2] J. L. Wyatt Jr., *et al*, "The first two years of the MIT vision chip project," Tech. Rep. VLSI Memo 90-605, Massachusetts Institute of Technology, October 1990.

[3] W. D. Hillis, *The Connection Machine.* New Haven, CT: MIT Press, 1985.

[4] A. Lumsdaine, J. Wyatt, and I. Elfadel, "Nonlinear analog networks for image smoothing and segmentation," in *International Symposium on Circuits and Systems*, (New Orleans, Louisiana), pp. 987–991, May 1990.

[5] L. W. Nagel, "SPICE2: A computer program to simulate semiconductor circuits," Tech. Rep. ERL M520, Electronics Research Laboratory Report, University of California, Berkeley, Berkeley, California, May 1975.

[6] H. C. Elman, *Iterative Methods for Large Sparse Nonsymmetric Systems of Linear Equations.* PhD thesis, Computer Science Dept., Yale University, New Haven, CT, 1982.

[7] P. Sonneveld, "CGS, a fast Lanczos-type solver for nonsymmetric linear systems," *SIAM J. Sci. Statist. Comput.*, vol. 10, pp. 36–52, 1989.

[8] R. Burch, K. Mayaram, J.-H. Chern, P. Yang, and P. Cox, "PGS and PLUCGS - Two new matrix solution techniques for general circuit simulation," in *International Conference on Computer Aided-Design*, (Santa Clara, California), pp. 408–411, November 1989.

[9] K. Mayaram, P. Yang, J. Chern, R. Burch, L. Arledge, and P. Cox, "A parallel block-diagonal preconditioned conjugate-gradient solution algorithm for circuit and device simulations," in *International Conference on Computer Aided-Design*, (Santa Clara, California), pp. 446–449, November 1990.

[10] L. M. Silveira, "Circuit simulation algorithms for massively parallel processors," Master's thesis, Massachusetts Institute of Technology, May 1990.

[11] D. M. Webber and A. Sangiovanni-Vincentelli, "Circuit simulation on the Connection Machine," in 24^{th} *ACM/IEEE Design Automation Conference*, June 1987.

[12] C. Tong, "The preconditioned conjugate gradient method on the Connection Machine," in *Proceedings of the Conference on Scientific Applications on the CM* (H. D. Simon, ed.), pp. 188–213, World Scientific, Singapore, September 1988.

[13] P. H. Worley, "Limits on parallelism in the numerical solution of linear partial differential equations," *SIAM J. Sci. Statist. Comput.*, vol. 12, pp. 1–35, January 1991.

[14] G. Golub and C. Van Loan, *Matrix Computations.* Baltimore: The Johns Hopkins University Press, 1989.

[15] A. Lumsdaine, M. Silveira, and J. White, "SIMLAB programmer's guide." Research Laboratory of Electronics, Massachusetts Institute of Technology. Unpublished, 1990.

[16] M. Silveira, A. Lumsdaine, and J. White, "SIMLAB users' guide." Research Laboratory of Electronics, Massachusetts Institute of Technology, 1990.

[17] Thinking Machines Corporation, Cambridge, Massachusetts, USA, *Connection Machine Model CM-2 Technical Summary*, May 1989. Technical Report-General TR 89-1.

[18] A. Lumsdaine, M. Silveira, and J. White, "CMVSIM programmer's guide." Research Laboratory of Electronics, Massachusetts Institute of Technology. Unpublished, 1990.

[19] A. Lumsdaine, M. Silveira, and J. White, "CMVSIM users' guide." Research Laboratory of Electronics, Massachusetts Institute of Technology, 1990.

[20] Thinking Machines Corporation, Cambridge, MA, *C* Programmer's Guide*, November 1990.

[21] Thinking Machines Corporation, Cambridge, MA, *C* User's Guide*, November 1990.

[22] A. Lumsdaine, J. Wyatt, and I. Elfadel, "Nonlinear analog networks for image smoothing and segmentation," *Journal of VLSI Signal Processing*, vol. 3, pp. 53–68, 1991.

4

Conjugate Direction Waveform Methods with Application to Semiconductor Device Transient Simulation

4.1 Introduction

The enormous computational expense and the growing importance of mixed circuit/device simulation, as well as the increasing availability of parallel computers, suggest that specialized, easily parallelized, algorithms be developed for transient simulation of MOS devices [1]. Recently, the easily parallelized waveform relaxation (WR) algorithm was shown to be a computationally efficient approach to device transient simulation [2]. However, the WR algorithm typically requires hundreds of iterations to achieve an accurate solution, which suggests that significant performance gains can still be realized by the application of methods for accelerating the convergence of the WR algorithm.

For linear algebra problems, conjugate-direction algorithms have enjoyed success as techniques for accelerating classical relaxation methods. Since the WR algorithm can in some sense be considered a function-space generalization of a linear algebra relaxation method, it seems only natural that conjugate-direction methods can be similarly generalized. Such a generalization can be rigorously analyzed with respect to asymptotic behavior by formulating the conjugate-direction method as a Galerkin method.

Beginning in the next section, a Galerkin method will be developed for solving an operator equation formulation of the linear time-varying initial-value problem. It is then shown that certain conjugate-direction methods iteratively generate the Galerkin approximations. The resulting methods are then combined with an operator Newton method in a hybrid scheme for solving the nonlinear initial-value problem. The semiconductor device transient simulation problem is described in Section 4.3. In Section 4.5, experimental results are presented which demonstrate that the conjugate-direction acceleration significantly reduces the computation time for device transient simulation.

4.2 Description of the Method

Consider the problem of numerically solving the linear time-varying initial value problem (IVP),

$$\begin{aligned} (\tfrac{d}{dt} + \boldsymbol{A}(t))\boldsymbol{x}(t) &= \boldsymbol{b}(t) \\ \boldsymbol{x}(0) &= \boldsymbol{x}_0, \end{aligned} \tag{4.1}$$

where $\boldsymbol{A}(t) \in \mathbb{R}^{N\times N}$, $\boldsymbol{b}(t) \in \mathbb{R}^N$ is a given right-hand side, and $\boldsymbol{x}(t) \in \mathbb{R}^N$ is the unknown vector to be computed over the simulation interval $t \in [0, T]$. There are several approaches to solving the IVP. The traditional numerical approach is to begin by discretizing (4.1) in time with an implicit integration rule (since large dynamical systems are typically stiff) and then solving the resulting matrix problem at each time step. 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 for each timestep.

A more effective approach to solving the IVP with a parallel computer is to decompose the problem at the ODE level. That is, the large system is decomposed into smaller subsystems, each of which is assigned to a single processor. The IVP is solved iteratively by solving the smaller IVP's for each subsystem, using fixed values from previous iterations for the variables from other subsystems. This dynamic iteration process is known as waveform relaxation (WR) or sometimes as the Picard-Lindelöf iteration [3].

In this section, conjugate-direction methods are considered for accelerating the classical dynamic iteration methods. The approach is to first convert the IVP to a system of second-kind Volterra integral equations by using a "dynamic preconditioner." Next, it is shown that the classical dynamic iteration methods are obtained by applying the Richardson iteration to the integral equation system. Finally, conjugate-direction methods are developed for accelerating the classical dynamic iteration methods. This development is approached by considering conjugate-direction methods as Galerkin methods.

4.2.1 Operator Equation Formulation

Let $\boldsymbol{A}(t) = \boldsymbol{M}(t) - \boldsymbol{N}(t)$ and consider the system of second kind Volterra integral equations given by

$$\boldsymbol{x}(t) - \boldsymbol{\Phi}_M(t,0)\boldsymbol{x}(0) - \int_0^t \boldsymbol{\Phi}_M(t,s)\boldsymbol{N}(s)\boldsymbol{x}(s)ds = \int_0^t \boldsymbol{\Phi}_M(t,s)\boldsymbol{b}(s)ds, \tag{4.2}$$

where $\boldsymbol{\Phi}_M$ is the state transition matrix [4] for the equation

$$\tfrac{d}{dt}\boldsymbol{x}(t) = \boldsymbol{M}(t)\boldsymbol{x}(t).$$

It is assumed throughout that $\boldsymbol{A}$, $\boldsymbol{M}$, and $\boldsymbol{N}$ are such that (4.1) and (4.2) each have a unique solution. A sufficient condition for this assumption is that $\boldsymbol{A}$, $\boldsymbol{M}$, and $\boldsymbol{N}$ be piecewise continuous; a weaker sufficient condition is that $\boldsymbol{A}$, $\boldsymbol{M}$, and $\boldsymbol{N}$ be measurable. Note that the solution $\boldsymbol{x}$ to (4.2) also satisfies (4.1). In some sense, (4.2) is obtained from (4.1) by the application of a "dynamic preconditioner" to both sides of (4.1). More precisely, this preconditioner, denoted $\mathcal{M}^{-1}$, is defined by:

$$(\mathcal{M}^{-1}\boldsymbol{x})(t) = \int_0^t \boldsymbol{\Phi}_M(t,s)\boldsymbol{x}(s)ds.$$

Informally, one can think of $\mathcal{M}^{-1}$ as being $(\frac{d}{dt} + \boldsymbol{M}(t))^{-1}$.

Equation (4.2) can be expressed as an operator equation over a space $\mathbb{H}$ as

$$(\boldsymbol{I} - \mathcal{K})\boldsymbol{x} = \boldsymbol{\psi}, \tag{4.3}$$

where $\mathbb{H} = \mathbb{L}_2([0,T],\mathbb{R}^N)$, $\boldsymbol{I} : \mathbb{H} \to \mathbb{H}$ is the identity operator, $\mathcal{K} : \mathbb{H} \to \mathbb{H}$ is defined by

$$\begin{aligned}(\mathcal{K}\boldsymbol{x})(t) &= \int_0^t \boldsymbol{K}_M(t,s)\boldsymbol{x}(s)ds \\ &= \int_0^t \boldsymbol{\Phi}_M(t,s)\boldsymbol{N}(s)\boldsymbol{x}(s)ds\end{aligned}$$

and $\boldsymbol{\psi} \in \mathbb{H}$ is given by

$$\boldsymbol{\psi}(t) = \boldsymbol{\Phi}_M(t,0)\boldsymbol{x}(0) + \int_0^t \boldsymbol{\Phi}_M(t,s)\boldsymbol{b}(s)ds.$$

The following are standard results (see, e.g., [5, 6]) which will be used in subsequent discussions of (4.3).

Lemma 4.2.1. If $\boldsymbol{M}$ and $\boldsymbol{N}$ are piecewise continuous (or measurable) then $\boldsymbol{\Phi}_M$ is measurable and hence $\boldsymbol{K}_M \in \mathbb{L}_2([0,T]\times[0,T],\mathbb{R}^{N\times N})$.

Lemma 4.2.2. If $\boldsymbol{K}_M \in \mathbb{L}_2([0,T]\times[0,T],\mathbb{R}^{N\times N})$, then the operator $\mathcal{K}$ has spectral radius zero.

Lemma 4.2.3. If $\boldsymbol{K}_M \in \mathbb{L}_2([0,T]\times[0,T],\mathbb{R}^{N\times N})$, then the operator $\mathcal{K}$ is compact.

Lemma 4.2.4. If $\boldsymbol{K}_M \in \mathbb{L}_2([0,T]\times[0,T],\mathbb{R}^{N\times N})$, then $\mathcal{K}^*$, the adjoint operator for $\mathcal{K}$, is given by

$$\begin{aligned}(\mathcal{K}^*\boldsymbol{x})(t) &= \int_t^T [\boldsymbol{K}_M(s,t)]^\dagger \boldsymbol{x}(s)ds \\ &= \int_t^T [\boldsymbol{\Phi}_M(s,t)\boldsymbol{N}(t)]^\dagger \boldsymbol{x}(s)ds,\end{aligned}$$

where superscript † denotes algebraic transposition.

Remark. It should be apparent from Lemma 4.2.4 that, in general, $\mathcal{K}$ is not self-adjoint. Therefore, attention is restricted to those conjugate-direction methods which are appropriate for non-self-adjoint operators.

4.2.2 Classical Dynamic Iteration Methods

The classical dynamic iteration is obtained by applying the Richardson iteration to the "preconditioned" problem (4.3):

$$\boldsymbol{x}^{k+1} = \mathcal{K}\boldsymbol{x}^k + \boldsymbol{\psi}. \tag{4.4}$$

This approach is known as the method of successive approximations, waveform relaxation, or the Picard-Lindelöf iteration [3, 6, 7, 8, 9].

Example. Let $\boldsymbol{M}(t) = 0$. Then $\boldsymbol{\Phi}_M = \boldsymbol{I}$ so that (4.2) becomes

$$\boldsymbol{x}(t) - \boldsymbol{x}(0) + \int_0^t \boldsymbol{A}(s)\boldsymbol{x}(s)ds = \int_0^t \boldsymbol{b}(s)ds.$$

The corresponding dynamic iteration is

$$\boldsymbol{x}^{k+1}(t) = \boldsymbol{x}(0) + \int_0^t \left(\boldsymbol{b}(s) - \boldsymbol{A}(s)\boldsymbol{x}^k(s)\right) ds$$

which is the familiar Picard iteration.

Example. Let $\boldsymbol{M}(t)$ be the diagonal part of $\boldsymbol{A}(t)$. Then (4.4) becomes the Jacobi WR algorithm where the following IVP is solved at each iteration k for each $x_i^{k+1}(t)$:

$$\begin{aligned}\left(\tfrac{d}{dt} + a_{ii}(t)\right) x_i^{k+1}(t) + \sum_{j\neq i} a_{ij}(t)x_j^k(t) &= b_i(t) \\ x_i(0) &= x_{0i}.\end{aligned}$$

As a direct consequence of Lemma 4.2.2, one has:

Theorem 4.2.5. Under the assumptions of Lemma 4.2.1, the method of successive approximations, defined in (4.4), converges.

Remark. Theorem 4.2.5 only provides a description of the asymptotic behavior of the dynamic iteration. In [3], Miekkala and Nevanlinna examine the dynamic iteration on the infinite interval. In that case, more intuition about the convergence rate of the dynamic iteration can be obtained, although the extent to which that intuition applies for a given dynamic iteration on a finite interval depends on the stiffness of the problem.

4.2.3 Accelerating Dynamic Iteration Methods

Another approach to solving (4.3) is to apply a Galerkin method to solving a variational formulation of the problem. This approach leads directly to the conjugate-direction methods. Galerkin methods have been well studied for second-kind Fredholm integral equations [6, 7], of which second-kind Volterra equations are a special case, but infrequently studied for second-kind Volterra equations in particular (see, however, [10]). With the conjugate-direction approach, instead of applying the Galerkin method over a space of polynomials or splines, as is typical, one applies the Galerkin method over a Krylov space generated by $(\boldsymbol{I} - \mathcal{K})$. The use of a Galerkin method over a Krylov space generated by $(\boldsymbol{I} - \mathcal{K})$ is discussed in [11] and [12] where the approach is called the method of moments (see also [13]).

The Galerkin Method

Let $\mathbb{X}$ and $\mathbb{Y}$ be Hilbert spaces and consider the operator equation

$$\mathcal{A}\boldsymbol{x} = \boldsymbol{b} \tag{4.5}$$

where $\boldsymbol{x} \in \mathbb{X}, \boldsymbol{b} \in \mathbb{Y}$ and $\mathcal{A} : \mathbb{X} \to \mathbb{Y}$ is a bounded injective operator.

Here, a Galerkin method is any scheme where the solution $\boldsymbol{x}$ (4.5) is computed by solving the problem in a sequence of finite-dimensional subspaces via the use of orthogonal projections. That is, one takes the subspaces $\mathbb{X}^n \subset \mathbb{X}$ and $\mathbb{Y}^n \subset \mathbb{Y}$ with $\dim \mathbb{X}^n = \dim \mathbb{Y}^n = n$ and requires the Galerkin approximation $\boldsymbol{x}^n$ to satisfy

$$\langle \boldsymbol{b} - \mathcal{A}\boldsymbol{x}^n, \boldsymbol{y} \rangle = 0 \quad \forall \boldsymbol{y} \in \mathbb{Y}^n. \tag{4.6}$$

In general, it is sufficient to satisfy (4.6) over some basis of $\mathbb{Y}^n$ by defining $\mathbb{X}^n = \operatorname{span}\{\boldsymbol{u}^0, \boldsymbol{u}^1, \ldots, \boldsymbol{u}^{n-1}\}$ and $\mathbb{Y}^n = \operatorname{span}\{\boldsymbol{v}^0, \boldsymbol{v}^1, \ldots, \boldsymbol{v}^{n-1}\}$, so that the solution $\boldsymbol{x}^n$ must satisfy

$$\langle \boldsymbol{b} - \mathcal{A}\boldsymbol{x}^n, \boldsymbol{v}^j \rangle = 0 \quad j = 0, 1, \ldots, n-1. \tag{4.7}$$

If $\boldsymbol{x}^n$ is taken to be

$$\boldsymbol{x}^n = \sum_{i=0}^{n-1} \gamma^i \boldsymbol{u}^i$$

then (4.7) generates a linear system of equations for $\{\gamma^i\}$:

$$\langle \mathcal{A} \sum_{i=0}^{n-1} \gamma^i \boldsymbol{u}^i, \boldsymbol{v}^j \rangle = \langle \boldsymbol{b}, \boldsymbol{v}^j \rangle.$$

The crucial question, of course, is whether or not a particular Galerkin method converges. To answer this, the following notion of convergence (which is standard for the Galerkin method [6, 14]) is used:

Definition 4.2.6. The Galerkin method is said to be *convergent* for the operator $\mathcal{A}$ if the following hold for every $\boldsymbol{b} \in \mathbb{Y}$,

1. The solution $\boldsymbol{x} \in \mathbb{X}$ to the original equation (4.5) exists and is unique.

2. Either:

    (a) There exists an index M such that for every $n \geq M$, the Galerkin equation (4.7) has a unique solution $\boldsymbol{x}^n$.

    (b) The approximate solutions $\boldsymbol{x}^n$ converge, i.e., $\boldsymbol{x}^n \rightarrow \boldsymbol{x}$ as $n \rightarrow \infty$.

    or

    (a) There exist indices M and N such that for every n, $M \leq n \leq N$, the Galerkin equation (4.7) has a unique solution $\boldsymbol{x}^n$.

    (b) The solution $\boldsymbol{x}^N = \boldsymbol{x}$.

The particular Galerkin method in which $\mathbb{Y} = \mathbb{X}$ and $\mathbb{Y}^n = \mathbb{X}^n$ is often called the Bubnov-Galerkin method. If $\mathcal{A}$ is positive definite in addition to being bounded and injective, it is well known that the Bubnov-Galerkin method is convergent for (4.5) [15]. Furthermore, if $\mathcal{A}$ is self-adjoint, the Galerkin approximations can be computed iteratively with the conjugate-gradient method (appropriately extended from $\mathbb{R}^N$ to $\mathbb{X}$, of course) [6].

For the problem under consideration, the operator $(\boldsymbol{I} - \mathcal{K})$ is not self-adjoint, yet one still seeks a conjugate-direction method appropriate for solving (4.3). Such methods can be derived by considering the Galerkin method where $\mathbb{Y} = \mathcal{A}(\mathbb{X})$ and $\mathbb{Y}^n = \mathcal{A}(\mathbb{X}^n)$. That is, $\boldsymbol{x}^n$ is required to satisfy

$$\langle \boldsymbol{b} - \mathcal{A}\boldsymbol{x}^n, \mathcal{A}\boldsymbol{u}^j \rangle = 0 \quad j = 0, 1, \ldots, n-1. \tag{4.8}$$

Note that this method can also be derived from a variational formulation of the problem. That is, instead of solving (4.5) directly, the following functional is minimized:

$$\phi(\boldsymbol{x}) = \|\boldsymbol{b} - \mathcal{A}\boldsymbol{x}\|^2,$$

where $\|\boldsymbol{x}\|^2 = \langle \boldsymbol{x}, \boldsymbol{x} \rangle$. Clearly, if (4.5) has a solution, the $\boldsymbol{x}$ which minimizes ϕ will also satisfy (4.5).

To be a minimizer of $\phi(\boldsymbol{x})$ over $\boldsymbol{x} \in \mathbb{X}^n$, the projection of the gradient of ϕ onto $\mathbb{X}^n$ must be zero, that is,

$$\langle \nabla\phi(\boldsymbol{x}^n), \boldsymbol{u}^j \rangle = 0 \quad j = 0, 1, \ldots, n-1. \tag{4.9}$$

An expression for the gradient, $\nabla\phi(\boldsymbol{x})$, can be obtained by straightforward calculation and is given in the following claim.

Claim 4.2.7.

$$\langle \nabla\phi(\boldsymbol{x}), \boldsymbol{y} \rangle = \langle \boldsymbol{b} - \mathcal{A}\boldsymbol{x}, \mathcal{A}\boldsymbol{y} \rangle$$

By observation it can be noted that (4.9) is equivalent to (4.8). This particular method is also known as the method of least squares [6, 15].

Now the main result can be stated.

Theorem 4.2.8. Let $\mathbb{X}$ be a Hilbert space and let $\mathcal{A} : \mathbb{X} \to \mathbb{X}$ be a bounded bijective linear operator. Let $\mathbb{X}^n \subset \mathbb{X}$ be a finite-dimensional subspace with $\mathbb{X}^n \subset \mathbb{X}^{n+1}$ for all $n \in \mathbb{N}$. If $\boldsymbol{x}$ is in the closure of $\mathbb{S} = \cup_{n=1}^{\infty} \mathbb{X}^n$, then the Galerkin method for (4.5) is convergent. Moreover, there exists the estimate

$$\|\boldsymbol{x} - \boldsymbol{x}^n\| \leq C\|\boldsymbol{b} - \mathcal{A}\boldsymbol{x}^n\| \tag{4.10}$$

for some constant C depending only on $\mathcal{A}$.

Proof. The conditions for the method to be convergent are verified. Let

$$\mathbb{X}^n = \text{span}\{\boldsymbol{u}^0, \boldsymbol{u}^1, \ldots \boldsymbol{u}^{n-1}\}.$$

Since $\mathcal{A}$ is bijective, the solution to (4.5) exists and is unique. Furthermore, there exists a constant $C = \|\mathcal{A}^{-1}\|$ such that

$$\|\boldsymbol{x}\| \leq C\|\mathcal{A}\boldsymbol{x}\|. \tag{4.11}$$

Case 1: It is first assumed that $\dim \mathbb{X}^n = n$ for all $n \in \mathbb{N}$. If the Galerkin equations (4.8) are not uniquely solvable for some n, then the homogeneous portion of (4.8) has a non-trivial solution, i.e., there exists a set of coefficients $\{\gamma^k\}$, not all zero, such that

$$\langle \sum_{k=0}^{n-1} \gamma^k \mathcal{A}\boldsymbol{u}^k, \mathcal{A}\boldsymbol{u}^j \rangle = 0 \quad j = 0, 1, \ldots, n-1.$$

A linear combination of $\boldsymbol{u}^j$'s is taken to obtain

$$\langle \sum_{k=0}^{n-1} \gamma^k \mathcal{A}\boldsymbol{u}^k, \sum_{j=0}^{n-1} \gamma^j \mathcal{A}\boldsymbol{u}^j \rangle = 0,$$

which contradicts the assumptions that $\mathcal{A}$ is injective and $\dim \mathbb{X}^n = n$. Therefore, (4.8) is solvable for all $n \geq 1$.

If $\boldsymbol{x} \in \operatorname{cl} \mathbb{S}$, then for any $\epsilon > 0$, there exists a $\boldsymbol{y} \in \mathbb{S}$ such that

$$\|\mathcal{A}(\boldsymbol{x} - \boldsymbol{y})\| \leq \frac{\epsilon}{C}.$$

Since $\boldsymbol{y} \in \mathbb{S}$, there must be an integer N such that $\boldsymbol{y} \in \mathbb{X}^N$. But in that case, one must also have for $n \geq N$

$$\|\mathcal{A}(\boldsymbol{x} - \boldsymbol{x}^n)\| \leq \|\mathcal{A}(\boldsymbol{x} - \boldsymbol{y})\| \leq \frac{\epsilon}{C},$$

because, from (4.9), $\boldsymbol{x}^N$ minimizes $\|\mathcal{A}(\boldsymbol{x} - \boldsymbol{y})\|$ for all $\boldsymbol{y} \in \mathbb{X}^N$ and, as $\mathbb{X}^n \supset \mathbb{X}^N$ for all $n \geq N$, the minimum does not increase for $n \geq N$. Therefore, by (4.11), $\|\boldsymbol{x} - \boldsymbol{x}^n\| \leq \epsilon$ for $n \geq N$. Thus, $\boldsymbol{x}^n \to \boldsymbol{x}$ as $n \to \infty$.

Case 2: Next, assume that $\dim \mathbb{S} = N$. Without loss of generality, one can take $\mathbb{S} = \operatorname{span}\{\boldsymbol{u}^0, \boldsymbol{u}^1, \ldots, \boldsymbol{u}^{N-1}\}$ and form $\mathbb{X}^n = \operatorname{span}\{\boldsymbol{u}^0, \boldsymbol{u}^1, \ldots, \boldsymbol{u}^{n-1}\}$. The Galerkin equations are then uniquely solvable for $n = 1, 2, \ldots, N$. If $\boldsymbol{x} \in \operatorname{cl} \mathbb{S}$, $\boldsymbol{x}$ can be expressed as

$$\boldsymbol{x} = \sum_{i=0}^{N-1} \alpha^i u^i.$$

Since $\boldsymbol{b} = \mathcal{A}\boldsymbol{x}$, (4.8) is equivalent to

$$\langle \mathcal{A}(\boldsymbol{x} - \boldsymbol{x}^N), \mathcal{A}\boldsymbol{u}^j \rangle = \langle \mathcal{A} \sum_{i=0}^{N-1} (\alpha^i - \gamma^i)\boldsymbol{u}^i, \mathcal{A}\boldsymbol{u}^j \rangle = 0 \quad j = 0, 1, \ldots, N-1.$$

In particular,

$$\langle \mathcal{A} \sum_{i=0}^{N-1} (\alpha^i - \gamma^i)\boldsymbol{u}^i, \mathcal{A} \sum_{i=0}^{N-1} (\alpha^i - \gamma^i)\boldsymbol{u}^i, \rangle = 0,$$

implying that $\alpha^i = \gamma^i$, $i = 0, \ldots, N-1$, i.e., that $\boldsymbol{x}^N = \boldsymbol{x}$.

The estimate (4.10) follows from (4.11):

$$\|\boldsymbol{x} - \boldsymbol{x}^n\| \leq C\|\mathcal{A}(\boldsymbol{x} - \boldsymbol{x}^n)\| = C\|\mathcal{A}\boldsymbol{x} - \mathcal{A}\boldsymbol{x}^n\|$$

so that

$$\|\boldsymbol{x} - \boldsymbol{x}^n\| \leq C\|\boldsymbol{b} - \mathcal{A}\boldsymbol{x}^n\|.$$

□

Remark. The case for which $\dim \mathbb{S} = \infty$ but $\dim \mathbb{X}^n \neq n$ for all n is handled by redefining the $\mathbb{X}^n$'s so that the assumption $\dim \mathbb{X}^n = n$ holds (see [15, p. 93]).

Corollary 4.2.9. The Galerkin method described in Theorem 4.2.8 is convergent for (4.3) in the space $\mathbb{H}$, with finite-dimensional subspaces $\mathbb{H}^n = \{\boldsymbol{\psi}, \mathcal{K}\boldsymbol{\psi}, \ldots \mathcal{K}^{n-1}\boldsymbol{\psi}\}$ for all $n \in \mathbb{N}$.

Proof. By the Riesz theory for compact operators, $(\boldsymbol{I} - \mathcal{K})$ is bounded and bijective since $\mathcal{K}$ is compact. Let $\mathbb{S} = \cup_{n=1}^{\infty} \mathbb{H}^n$. The recursive definition of $\mathbb{H}^n$ guarantees that

1. If $\dim \mathbb{S} = \infty$, then $\dim \mathbb{H}^n = n$ for all $n \in \mathbb{N}$, in which case the Galerkin equations are uniquely solvable for all $n \in \mathbb{N}$.

2. If $\dim \mathbb{S} = N$, then $\dim \mathbb{H}^n = n$ for all $1 \leq n \leq N$ and $\dim \mathbb{H}^n = N$ for all $n > N$.

To show that $\boldsymbol{x} \in \operatorname{cl} \mathbb{S}$, note that

$$\boldsymbol{x} = (\boldsymbol{I} - \mathcal{K})^{-1}\boldsymbol{\psi} = \sum_{j=0}^{\infty} \mathcal{K}^j \boldsymbol{\psi}$$

where the Neumann series for $(\boldsymbol{I} - \mathcal{K})^{-1}$ converges since the spectral radius of $\mathcal{K}$ is zero. Since the sum $\sum_{j=0}^{\infty} \mathcal{K}^j\boldsymbol{\psi}$ is clearly contained in $\operatorname{cl}\operatorname{span}\{\boldsymbol{\psi}, \mathcal{K}\boldsymbol{\psi}, \mathcal{K}^2\boldsymbol{\psi}, \ldots\}$, then $\boldsymbol{x} \in \operatorname{cl}\operatorname{span}\{\boldsymbol{u}^0, \boldsymbol{u}^1, \ldots\}$. □

Remark. Corollary 4.2.9 can be applied to the case where $\mathcal{A} = \mathcal{J} - \mathcal{K}$, provided $\mathcal{J}$ has a bounded inverse and the spectral radius of $\mathcal{J}^{-1}\mathcal{K}$ is strictly less than unity. In that case, $\mathcal{J}^{-1}$ can be applied to the system to obtain

$$(\boldsymbol{I} - \mathcal{J}^{-1}\mathcal{K})\boldsymbol{x} = \mathcal{J}^{-1}\boldsymbol{\psi}.$$

Corollary 4.2.9 is then applied since $\mathcal{J}^{-1}\mathcal{K}$ is compact.

Iterative Algorithms

Various iterative algorithms exist which can be used to implement the Galerkin method described in Corollary 4.2.9. The foremost of these in the linear algebra setting are the generalized minimum residual algorithm (GMRES) [16] and the generalized conjugate residual algorithm (GCR) [17]. The GMRES and GCR algorithms can be adapted

Algorithm 4.2.1 (WGMRES).

```
Set $r^0 = b - \mathcal{A}x^o$, $\beta = \|r^0\|$, and $v^0 = r^0/\beta$
For $k = 1, 2, \ldots$ until $\langle r^k, r^k \rangle < \epsilon$,
    $h_{j,k} = \langle \mathcal{A}v^j, v^k \rangle$, $i = 1, 2, \ldots, k$
    $\hat{v}^{k+1} = \mathcal{A}v^k - \sum_{j=1}^k h_{j,k} v^j$
    $h_{k+1,k} = \|\hat{v}^{k+1}\|$
    $v^{k+1} = \hat{v}^{k+1}/h_{k+1,k}$
Set $x^k = x^0 + V^k y^k$
    Here, $y^k$ minimizes $\|\beta e_1 - \bar{H}^k y^k\|$ where
        $\bar{H}^k$ is the $(k+1) \times k$ matrix with nonzero entries $h_{i,j}$,
        $V^k = [v^1, \ldots, v^k]$, and
        $e_1 = [1, 0, \ldots, 0]^T$
```

Algorithm 4.2.2 (WGCR).

```
Set $p^0 = r^0 = b - \mathcal{A}x^o$
For $k = 0, 1, \ldots$ until $\langle r^k, r^k \rangle < \epsilon$
    $\alpha = \frac{\langle \mathcal{A}p^k, r^k \rangle}{\langle \mathcal{A}p^k, \mathcal{A}p^k \rangle}$
    $x^{k+1} = x^k + \alpha p^k$
    $r^{k+1} = r^k - \alpha \mathcal{A}p^k$
    $p^{k+1} = r^{k+1} + \sum_{j=0}^k \beta_k^j p^j$
        where $\{\beta_k^j\}$ are chosen so that
        $\langle \mathcal{A}p^{k+1}, \mathcal{A}p^j \rangle = 0$ for $0 \leq j \leq k$
```

quite readily to the space $\mathbb{H}$ instead of $\mathbb{R}^N$ and are shown in Algorithm 4.2.1 and Algorithm 4.2.2, respectively. The function-space versions of these algorithms are referred to as WGMRES and WGCR (waveform GMRES and waveform GCR) to distinguish them from their linear algebra counterparts.

The two fundamental operations in Algorithm 4.2.1 and Algorithm 4.2.2 are the operator-function product $\mathcal{A}p$ and the inner product $\langle \cdot, \cdot \rangle$. When solving (4.3) in the space $\mathbb{H}$, these operations are as follows:

Operator-Function Product: To calculate $w \equiv (I - \mathcal{K})p$:

1. Solve the IVP

$$\begin{aligned} (\tfrac{d}{dt} + M(t))y(t) &= N(t)p(t) \\ y(0) &= p_0 = 0 \end{aligned}$$

for $\boldsymbol{y}(t)$, $t \in [0,T]$; this gives $\boldsymbol{y} = \mathcal{K}\boldsymbol{p}$.

2. Set $\boldsymbol{w} = \boldsymbol{p} - \boldsymbol{y}$

Inner Product: The inner product $\langle \boldsymbol{x}, \boldsymbol{y} \rangle$ is given by

$$\langle \boldsymbol{x}, \boldsymbol{y} \rangle = \sum_{i=1}^{N} \int_0^T x_i(t) y_i(t) dt.$$

Remark. Step 1 of the operator-function product is equivalent to one step of the classical dynamic iteration. Hence, the WGMRES and WGCR algorithms can be regarded as being accelerations of the classical dynamic iterations.

The conjugate-direction methods are just as amenable to parallel implementation as the classical dynamic iteration methods. As already discussed, one step of the dynamic iteration is embedded within the operator-function product. Moreover, the inner product is accomplished by N separate integrations of the pointwise product $x_i(t)y_i(t)$, which can be performed in parallel, followed by a global sum of the results.

Convergence Properties

The convergence result in Corollary 4.2.9 only provides an asymptotic estimate. A stronger convergence result can be obtained by making suitable assumptions about $\mathcal{K}$; in particular, it is assumed that $\mathcal{K}$ is a contraction.

Definition 4.2.10. An operator $\mathcal{A} : \mathbb{X} \to \mathbb{X}$ is said to be a *contraction* on $\mathbb{X}$ iff for all $\boldsymbol{x} \in \mathbb{X}$,

$$\|\mathcal{A}\boldsymbol{x}\| \leq \gamma \|\boldsymbol{x}\|$$

for some $\gamma \in [0,1)$.

The assumption that $\mathcal{K}$ be a contraction may seem unnecessarily strong; however, note that under the assumptions of Lemma 4.2.1, $\mathcal{K}$ is *always* a contraction over some interval $[0,T]$.

Lemma 4.2.11. Under the assumptions of Lemma 4.2.1, there exists an interval $[0,T]$ such that $\mathcal{K}$ is a contraction on $\mathbb{L}_2([0,T], \mathbb{R}^N)$.

Proof. By assumption, $\boldsymbol{K}_M(t,s) = \boldsymbol{\Phi}_M(t,s)\boldsymbol{N}(s)$ is measurable, so that

$$\|\boldsymbol{K}\|^2 \leq \int_0^T \int_0^T \boldsymbol{K}(t,s) ds dt \leq T^2 C_K^2$$

where $C_K = \text{ess.sup}_{s,t \in [0,T]} \boldsymbol{K}(t,s)$. But since

$$\|\mathcal{K}\boldsymbol{x}\| \leq \|\boldsymbol{K}\| \; \|\boldsymbol{x}\|$$

$T < \frac{\gamma}{C_K}$ can be chosen so that

$$\|\mathcal{K}\boldsymbol{x}\| \leq \gamma \|\boldsymbol{x}\|.$$

□

If $\mathcal{K}$ is a contraction, one can formulate the following result for the method of successive approximations.

Theorem 4.2.12. Let the system of (4.4) be such that $\mathcal{K}$ is a contraction, i.e., $\|\mathcal{K}\boldsymbol{x}\| \leq \gamma\|\boldsymbol{x}\|$ for some $\gamma \in [0,1)$. Then, the method of successive approximations, defined in (4.4), converges. Moreover, there exist the estimates

$$\|\boldsymbol{x}^{k+1} - \boldsymbol{x}^k\| \leq \gamma\|\boldsymbol{x}^k - \boldsymbol{x}^{k-1}\|$$

and

$$\|\boldsymbol{r}^k\| \leq \gamma^k\|\boldsymbol{r}^0\|.$$

Proof. Using (4.4),

$$\begin{aligned} \|\boldsymbol{x}^{k+1} - \boldsymbol{x}^k\| &= \|\mathcal{K}\boldsymbol{x}^k + \boldsymbol{\psi} - (\mathcal{K}\boldsymbol{x}^{k-1} + \boldsymbol{\psi})\| \\ &= \|\mathcal{K}(\boldsymbol{x}^k - \boldsymbol{x}^{k-1})\| \\ &\leq \gamma\|(\boldsymbol{x}^k - \boldsymbol{x}^{k-1})\|. \end{aligned}$$

Using $\|\boldsymbol{x}^{k+1} - \boldsymbol{x}^k\| = \|\boldsymbol{r}^k\|$, the second estimate follows from induction on the first. □

If $\mathcal{K}$ is a contraction, one can formulate similar results for WGCR.

Theorem 4.2.13. Let the system of (4.4) be such that $\mathcal{K}$ is a contraction, i.e., $\|\mathcal{K}\boldsymbol{x}\| \leq \gamma\|\boldsymbol{x}\|$ for some $\gamma \in [0,1)$. Then, the waveform generalized conjugate residual algorithm, Algorithm 4.2.2, converges. Moreover, there exists the estimate

$$\|\boldsymbol{r}^k\| \leq \min_{q^k \in \mathbb{P}^k} \|q^k(\boldsymbol{I} - \mathcal{K})\| \, \|\boldsymbol{r}^0\| \leq \left[1 - \frac{(1-\gamma)^2}{(1+\gamma)^2}\right]^{\frac{k}{2}} \|\boldsymbol{r}^0\|.$$

Proof. The result can be obtained by following the analogous proof for algebraic GCR [17, p. 40]. □

To demonstrate the convergence properties of WGCR, the results of three numerical experiments are presented. The three examples are of the 2×2 system

$$\tfrac{d}{dt}\boldsymbol{x}(t) + \boldsymbol{A}\boldsymbol{x}(t). \tag{4.12}$$

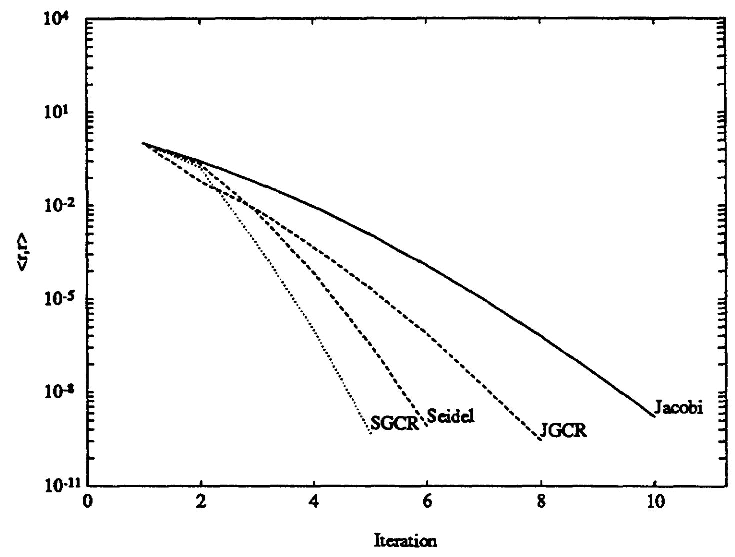

Figure 4-1: Convergence comparison between WR and WGCR for the 2×2 system (4.12). In this example, A has eigenvalues $\{1,10\}$ and $||r||^2$ is plotted against iteration number.

In the first experiment, A has eigenvalues $\{1,10\}$, in the second A has eigenvalues $\{1,100\}$, and in the third, A has eigenvalues $\{1,1000\}$. The plots shown in Figures 4-1 – 4-3 compare convergence of Jacobi WR (JWR), Seidel WR (SWR), Jacobi-preconditioned WGCR (JGCR), and Seidel-preconditioned WGCR (SGCR).

4.2.4 Hybrid Methods for Nonlinear Systems

Many interesting applications are not necessarily described by a linear system of ODE's, but rather by a nonlinear system of ODE's:

$$\begin{aligned} \tfrac{d}{dt}\boldsymbol{x}(t)+\boldsymbol{F}(\boldsymbol{x}(t),t) &= 0 \\ \boldsymbol{x}(0) &= \boldsymbol{x}_0. \end{aligned} \tag{4.13}$$

To solve (4.13), Newton's method is applied directly to the nonlinear ODE system (in a process sometimes referred to as the waveform Newton method (WN) [18]) to obtain the following iteration:

$$\begin{aligned} \left(\tfrac{d}{dt}+\boldsymbol{J}_F(\boldsymbol{x}^m)\right)\boldsymbol{x}^{m+1} &= \boldsymbol{J}_F(\boldsymbol{x}^m)\boldsymbol{x}^m - \boldsymbol{F}(\boldsymbol{x}^m) \\ \boldsymbol{x}^{m+1}(0) &= \boldsymbol{x}_0. \end{aligned} \tag{4.14}$$

Here, $\boldsymbol{J}_F$ is the Jacobian of $\boldsymbol{F}$. Note that (4.14) is a linear time-varying IVP to be solved for $\boldsymbol{x}^{m+1}$, which can be accomplished with a waveform conjugate direction method. The resulting operator Newton/conjugate-direction algorithm is shown in Algorithm 4.2.3; note that the method is in the class of hybrid Krylov methods [19].

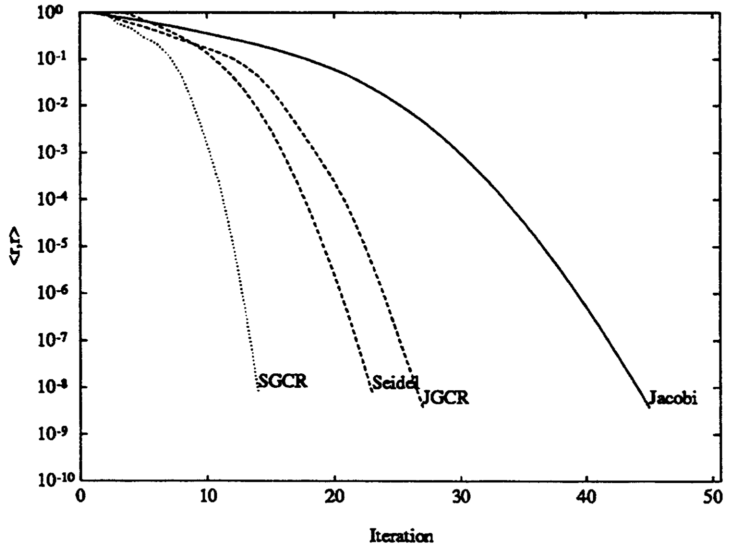

Figure 4-2: Convergence comparison between WR and WGCR for the 2×2 system (4.12). In this example, A has eigenvalues $\{1, 100\}$ and $\|r\|^2$ is plotted against iteration number.

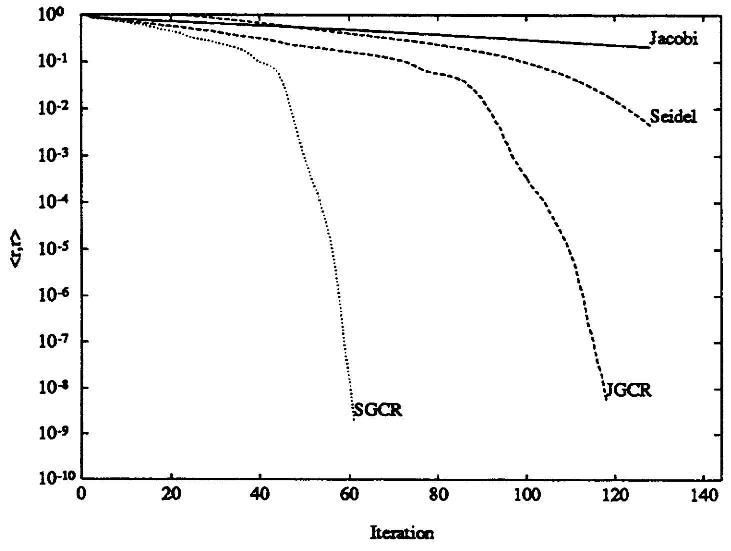

Figure 4-3: Convergence comparison between WR and WGCR for the 2×2 system (4.12). In this example, A has eigenvalues $\{1, 1000\}$ and $\|r\|^2$ is plotted against iteration number.

Algorithm 4.2.3 (Nonlinear GMRES/GCR).

```
Pick $x^0$, $\epsilon^0$, $\nu < 1$
For $m = 0, 1, \ldots$ until $\langle r^m, r^m \rangle < \phi$,
    Linearize (4.13) to form (4.14)
    Solve (4.14) with Algorithm 4.2.1 or Algorithm 4.2.2 using $\epsilon^m$
    Update $x^{m+1}$ and $r^{m+1}$
    Set $\epsilon^{m+1} = \epsilon^m \cdot \nu$
```

4.3 Device Transient Simulation

A device is assumed to be governed by the Poisson equation, and the electron and hole continuity equations:

$$\begin{aligned}
\frac{\epsilon kT}{q}\nabla^2 u + q(p - n + N_D - N_A) &= 0 \\
\nabla \cdot \boldsymbol{J}_n - q\left(\frac{\partial n}{\partial t} + R\right) &= 0 \\
\nabla \cdot \boldsymbol{J}_p + q\left(\frac{\partial p}{\partial t} + R\right) &= 0
\end{aligned}$$

where u is the normalized electrostatic potential, 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 electronic charge, and ϵ is the dielectric permittivity [20, 21].

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

$$\begin{aligned}
\boldsymbol{J}_n &= -qD_n(n\,\nabla u - \nabla n) \\
\boldsymbol{J}_p &= -qD_p(p\,\nabla u + \nabla p)
\end{aligned}$$

where D_n and D_p are the diffusion coefficients, which are assumed here to be related to the electron and hole mobilities by the Einstein relations, that is $D = \frac{kT}{q}\mu$. $\boldsymbol{J}_n$ and $\boldsymbol{J}_p$ are typically eliminated from the continuity equations using the drift-diffusion approximations, leaving a differential-algebraic system of three equations in three unknowns, u, n, and p.

Given a rectangular mesh that covers a two-dimensional slice of a MOSFET, a common approach to spatially discretizing the device equations is to use a finite-difference formula to discretize the Poisson equation, and an exponentially-fit finite-difference formula to discretize the continuity equations (the Scharfetter-Gummel method) [22]. On an

N-node rectangular mesh, the spatial discretization yields a differential-algebraic system of $3N$ equations in $3N$ unknowns denoted by

$$\boldsymbol{f}_1(\boldsymbol{u}(t),\boldsymbol{n}(t),\boldsymbol{p}(t)) = 0 \tag{4.15}$$

$$\boldsymbol{f}_2(\boldsymbol{u}(t),\boldsymbol{n}(t),\boldsymbol{p}(t)) = \tfrac{d}{dt}\boldsymbol{n}(t) \tag{4.16}$$

$$\boldsymbol{f}_3(\boldsymbol{u}(t),\boldsymbol{n}(t),\boldsymbol{p}(t)) = \tfrac{d}{dt}\boldsymbol{p}(t) \tag{4.17}$$

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, respectively. Here, $\boldsymbol{f}_1,\boldsymbol{f}_2,\boldsymbol{f}_3 : \mathbb{R}^{3N} \to \mathbb{R}^N$ are specified component-wise as

$$\begin{aligned}
f_{1_i}(u_i,n_i,p_i,u_j) &= \frac{\epsilon kT}{q}\sum_j \frac{d_{ij}}{L_{ij}}(u_i-u_j) - qA_i\,(p_i - n_i + N_D - N_A)\\
f_{2_i}(u_i,n_i,u_j,n_j) &= \frac{D_n}{A_i}\sum_j \frac{d_{ij}}{L_{ij}}\Big[n_jB(u_j-u_i) - n_iB(u_i-u_j)\Big] - R_i\\
f_{3_i}(u_i,p_i,u_j,p_j) &= \frac{D_p}{A_i}\sum_j \frac{d_{ij}}{L_{ij}}\Big[p_jB(u_i-u_j) - p_iB(u_j-u_i)\Big] - R_i.
\end{aligned}$$

The sums above are taken over the four nodes adjacent to node i (north, south, east, and west), 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 $B(v) = v/(e^v - 1)$ is the Bernoulli function, used to exponentially fit potential variation to electron concentration variation.

On the order of a thousand mesh nodes are typically needed to accurately represent a 2-D slice of a MOS transistor, so that simulating a circuit where even a few transistors are treated by numerically solving the device equations leads to an enormous coupled system of algebraic and differential equations.

The standard approach used to solve the differential-algebraic system generated by spatial discretization of the device equations is to discretize the d/dt terms with a low-order implicit integration method such as the second-order backward difference formula. The result is a sequence of nonlinear algebraic systems in $3N$ unknowns, each of which can be solved with some variant of Newton's method and/or relaxation [23]. Another approach is to apply relaxation directly to the differential-algebraic equation system with a WR algorithm [2, 24], as given in Algorithm 4.3.1.

In the present approach, the hybrid Krylov method described in Section 4.2.4 is applied to (4.15)–(4.17) to obtain the following IVP at each Newton iteration m:

$$\begin{bmatrix} 0 \\ \frac{d}{dt}\boldsymbol{n}^{m+1} \\ \frac{d}{dt}\boldsymbol{p}^{m+1} \end{bmatrix} + \begin{bmatrix} \boldsymbol{J}_{f_{11}} & \boldsymbol{J}_{f_{12}} & \boldsymbol{J}_{f_{13}} \\ \boldsymbol{J}_{f_{21}} & \boldsymbol{J}_{f_{22}} & \boldsymbol{J}_{f_{23}} \\ \boldsymbol{J}_{f_{31}} & \boldsymbol{J}_{f_{32}} & \boldsymbol{J}_{f_{33}} \end{bmatrix} \begin{bmatrix} \boldsymbol{u}^{m+1} \\ \boldsymbol{n}^{m+1} \\ \boldsymbol{p}^{m+1} \end{bmatrix}$$

Algorithm 4.3.1 (WR for Device Simulation).

```
guess $u^0, n^0, p^0$ waveforms at all nodes
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:
      $f_{1_i}(u_i^{k+1}, n_i^{k+1}, p_i^{k+1}, u_j^k) = 0$
      $f_{2_i}(u_i^{k+1}, n_i^{k+1}, u_j^k, n_j^k) = \frac{d}{dt} n_i^{k+1}$
      $f_{3_i}(u_i^{k+1}, p_i^{k+1}, u_j^k, p_j^k) = \frac{d}{dt} p_i^{k+1}$
```

$$= \begin{bmatrix} J_{f_{11}} & J_{f_{12}} & J_{f_{13}} \\ J_{f_{21}} & J_{f_{22}} & J_{f_{23}} \\ J_{f_{31}} & J_{f_{32}} & J_{f_{33}} \end{bmatrix} \begin{bmatrix} u^m \\ n^m \\ p^m \end{bmatrix} - \begin{bmatrix} f_1(u^m, n^m, p^m) \\ f_2(u^m, n^m, p^m) \\ f_3(u^m, n^m, p^m) \end{bmatrix}$$

$$\begin{bmatrix} u^{m+1}(0) \\ n^{m+1}(0) \\ p^{m+1}(0) \end{bmatrix} = \begin{bmatrix} u_0 \\ n_0 \\ p_0 \end{bmatrix}.$$

4.4 Implementation

The nonlinear WGMRES and WGCR algorithms described in Section 4.2 were implemented in Mark Reichelt's *WORDS* program, a WR-based device transient simulation program. In addition, a waveform-relaxation-Newton algorithm (WRN) and a waveform conjugate-gradient-squared algorithm (WCGS) were implemented [9, 25]. As previously mentioned, the operator-waveform product has embedded within it one step of traditional waveform relaxation. Therefore, the operator-waveform product routine need only make a function call to the WR routine already implemented within *WORDS* — the preconditioning scheme inherent to the WR routine will automatically be used by the conjugate-direction method as well. In this section, the main WR routine already contained within *WORDS* is described [2] and the modifications made to that routine to support the conjugate direction methods are developed.

4.4.1 Main WR Routine

As reported in [2], the node-by-node Gauss-Jacobi WR algorithm as given in Algorithm 4.3.1 will typically require many hundreds (or even thousands) of iterations to converge, severely limiting the efficiency of WR-based device simulation. Moreover, assigning each node to a separate processor in a parallel implementation would require on the order of a thousand processors or more (the number of nodes necessary for accurate device simulation). Since the WR algorithm is more suited to a coarse-grained MIMD type of architecture, such a fine-grained division of the problem is not necessary or desirable. Instead of the node-by-node approach, *WORDS* collects groups of mesh nodes into blocks and solves the nodes in each block simultaneously. The block WR algorithm is similar to node-by-node WR, except that the system of equations for each node i in Algorithm 4.3.1 is replaced by a larger system of equations representing the systems of equations for all the nodes contained in block I. In each WR iteration, the solution $\boldsymbol{x}^k(t) = [\boldsymbol{x}_1(t), \ldots, \boldsymbol{x}_N(t)]^\dagger$ is computed by solving the equation system for $\boldsymbol{x}_I(t)$, the vector of $\boldsymbol{u}$, $\boldsymbol{n}$, and $\boldsymbol{p}$ waveforms of the nodes in block I, while holding the waveforms of neighboring mesh nodes in adjacent blocks fixed. This approach of blocking for WR is typical in WR-based circuit simulators [9].

Using blocks of nodes typically produces faster WR convergence; however, the computational expense of directly solving the system of equations for blocks of nodes is higher than that for solving the smaller system of equations for a single node. Furthermore, the same timepoints are used for all nodes within a block, so that a block algorithm cannot take advantage of multirate behavior within a block. The challenge is to find a blocking which covers the device mesh in relatively few easy-to-solve blocks and groups tightly-coupled nodes together, but that does not group nodes together which are expected to change at different rates.

The blocking scheme used by *WORDS* is to group together the nodes according to the vertical lines of the discretization mesh — this has been shown to be a particularly effective blocking strategy for MOSFET simulation [2] for the following reasons:

1. The vertical lines cross the oxide-silicon interface, a traditional source of numerical difficulty with a 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 block-based problems produce block-tridiagonal matrices which are easily solved.

3. Each vertical line in the channel of a MOSFET has both ends pinned by the gate and substrate contacts, so that the line's solution correctly accounts for these contacts and directly captures the nonlinear electric field dependence governing surface depletion and channel width from the very first WR iteration.

4. The vertical line blocking allows the WR algorithm to take advantage of the multirate behavior resulting from the horizontal distance between the contacts. For example, if the drain voltage of a MOSFET is increased while the source is held steady, 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.

The main WR routine in the *WORDS* program uses a red/black block Gauss-Seidel scheme, with vertical mesh-line blocks. In the main WR routine, the equations governing nodes in each block are solved simultaneously using the second-order backward-difference formula. The implicit algebraic systems generated 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.

4.4.2 Operator Waveform Product

Although the operator waveform product has one step of traditional WR embedded within it, the main WR routine within *WORDS* is nonlinear WR, so some small modifications are necessary to compute the linearized WR step as required by the operator waveform product.

For reasons of clarity, the methods for computing the linearized operator-waveform product are developed by considering the nonlinear IVP (4.13):

$$\begin{aligned} \tfrac{d}{dt}\boldsymbol{x}(t) &= \boldsymbol{F}(\boldsymbol{x}(t),t) \\ \boldsymbol{x}(0) &= \boldsymbol{x}_0 \end{aligned}$$

instead of the device simulation equations. The waveform Newton algorithm (4.14) applied to solving the nonlinear IVP can be written as:

$$\begin{aligned} \tfrac{d}{dt}\boldsymbol{x}^{m+1} + \boldsymbol{J_F}(\boldsymbol{x}^m)\boldsymbol{x}^{m+1} &= \boldsymbol{J_F}(\boldsymbol{x}^m)\boldsymbol{x}^m - \boldsymbol{F}(\boldsymbol{x}^m) \\ \boldsymbol{x}^{m+1}(0) &= \boldsymbol{x}_0 \end{aligned}$$

From (4.2), the preconditioned linear IVP is given by

$$\boldsymbol{x}^{m+1}(t) - \boldsymbol{\Phi}_M(t,0)\boldsymbol{x}^{m+1}(0) - \int_0^t \boldsymbol{\Phi}_M(t,s)\boldsymbol{N}(s)\boldsymbol{x}^{m+1}(s)ds =$$
$$\int_0^t \boldsymbol{\Phi}_M(t,s)\left(\boldsymbol{J_F}(\boldsymbol{x}^m(s),s)\boldsymbol{x}^m(s) - \boldsymbol{F}(\boldsymbol{x}^m(s),s)\right)ds,$$

or, in operator equation form, as

$$\left(\boldsymbol{I} - \mathcal{K}\right)\boldsymbol{x}^{m+1} = \boldsymbol{\psi}.$$

Equivalently, the result of the operation $\mathcal{K}\boldsymbol{x}$ can be expressed as being the vector $\tilde{\boldsymbol{x}}$ which satisfies the IVP

$$\begin{aligned} \left(\tfrac{d}{dt} + \boldsymbol{M}(\boldsymbol{x}(t),t)\right)\tilde{\boldsymbol{x}}(t) &= \boldsymbol{N}(\boldsymbol{x}(t),t)\tilde{\boldsymbol{x}}(t) \\ \tilde{\boldsymbol{x}}(0) &= \boldsymbol{x}_0 \end{aligned} \tag{4.18}$$

(recall that $\boldsymbol{J}_F(\boldsymbol{x}(t),t) = \boldsymbol{M}(\boldsymbol{x}(t),t) - \boldsymbol{N}(\boldsymbol{x}(t),t)$).

In the following discussion, three different approaches are examined for calculating the operator-waveform product product $(\boldsymbol{I} - \mathcal{K})\,\boldsymbol{x}$, or more specifically for calculating $\mathcal{K}\boldsymbol{x}$. The methods require increasing levels of modification to the main WR routine within *WORDS*.

Matrix-Free Approach (Level One)

The simplest means of obtaining an approximation to the linearized operator waveform product is to use a matrix-free approach. In this approach, $\mathcal{K}$ is approximated by

$$\mathcal{K}\boldsymbol{p} \approx \mathcal{W}(\boldsymbol{x}+\boldsymbol{p}) - \mathcal{W}(\boldsymbol{x})$$

where $\mathcal{W}(\boldsymbol{x})$ is taken to mean one iteration of the WR routine within *WORDS*, using $\boldsymbol{x}$ as the input.

The matrix-free approximation turns out to not be a very good approximation in practice. This is due to more than the effects of the perturbation due to $\boldsymbol{p}$. To see why, consider a 2×2 example:

$$\begin{aligned} \tfrac{d}{dt}x_1 + f_1(x_1,x_2) &= 0 \\ \tfrac{d}{dt}x_2 + f_2(x_1,x_2) &= 0 \end{aligned} \tag{4.19}$$

Let $\tilde{x}_1$ and $\tilde{x}_2$ be the output of one iteration of the Gauss-Seidel WR routine applied to solving (4.19). Then $\tilde{\boldsymbol{x}} = [\tilde{x}_1\ \tilde{x}_2]^\dagger$ satisfies

$$\begin{aligned} \tfrac{d}{dt}\tilde{x}_1 + f_1(\tilde{x}_1,x_2) &= 0 \\ \tfrac{d}{dt}\tilde{x}_2 + f_2(\tilde{x}_1,\tilde{x}_2) &= 0 \end{aligned} \tag{4.20}$$

Let $\tilde{\boldsymbol{y}}$ be the output of one iteration of the Gauss-Seidel WR routine applied to solving (4.19) with $\boldsymbol{y}$ as the input. Then

$$\begin{aligned} \tfrac{d}{dt}\tilde{y}_1 + f_1(\tilde{y}_1,y_2) &= 0 \\ \tfrac{d}{dt}\tilde{y}_2 + f_2(\tilde{y}_1,\tilde{y}_2) &= 0 \end{aligned} \tag{4.21}$$

Define $\boldsymbol{p} = \boldsymbol{y} - \boldsymbol{x}$ and $\tilde{\boldsymbol{p}} = \tilde{\boldsymbol{y}} - \tilde{\boldsymbol{x}}$. From (4.20) and (4.21), $\tilde{\boldsymbol{p}}$ satisfies (to first approximation):

$$\begin{aligned} \tfrac{d}{dt}\tilde{p}_1 + \tfrac{\partial f_1(\tilde{x}_1,x_2)}{\partial x_1}\tilde{p}_1 + \tfrac{\partial f_1(\tilde{x}_1,x_2)}{\partial x_2}p_2 &= 0 \\ \tfrac{d}{dt}\tilde{p}_2 + \tfrac{\partial f_2(\tilde{x}_1,\tilde{x}_2)}{\partial x_1}\tilde{p}_1 + \tfrac{\partial f_2(\tilde{x}_1,\tilde{x}_2)}{\partial x_2}\tilde{p}_2 &= 0 \end{aligned} \tag{4.22}$$

However, for $\tilde{\boldsymbol{p}}$ to be the result of the operator-waveform product within the conjugate-direction method, it should satisfy:

$$\begin{aligned} \frac{d}{dt}\tilde{p}_1 + \frac{\partial f_1(x_1,x_2)}{\partial x_1}\tilde{p}_1 + \frac{\partial f_1(x_1,x_2)}{\partial x_2}\tilde{p}_2 &= 0 \\ \frac{d}{dt}\tilde{p}_2 + \frac{\partial f_2(x_1,x_2)}{\partial x_1}\tilde{p}_1 + \frac{\partial f_2(x_1,x_2)}{\partial x_2}\tilde{p}_2 &= 0 \end{aligned} \tag{4.23}$$

Comparison of (4.22) with (4.23) shows that the evaluation point of the Jacobian is incorrect with the matrix-free approach.

The 2×2 example is easily generalized to N dimensions:

$$\mathcal{K}\boldsymbol{p} = \mathcal{W}(\boldsymbol{x}+\boldsymbol{p}) - \mathcal{W}(\boldsymbol{x}) + \mathcal{O}(\|\mathcal{W}(\boldsymbol{x}) - \boldsymbol{x}\|)$$

This inaccuracy is present with any nontrivial decomposition of the problem (e.g., Gauss-Jacobi will have this same inaccuracy). This inaccuracy is a function of $\|\tilde{\boldsymbol{x}} - \boldsymbol{x}\|$ and hence cannot be made smaller by scaling $\boldsymbol{p}$ as is typically done with matrix-free methods [19].

Relaxation Newton Approach (Level Two)

An improvement to the matrix-free approach is to instead use one step of a relaxation-Newton method. This approach attempts to correct the problems in the level one method regarding the evaluation points of the Jacobian. This approach is still approximate, although experiments comparing the performance of the level two method with the level one method indicate that the level two approach provides a much better approximation.

Consider one step of the Jacobi-Newton waveform method, in which the IVP

$$\begin{aligned} \left(\frac{d}{dt} + \boldsymbol{D}(\boldsymbol{x})\right)\tilde{\boldsymbol{x}} &= \boldsymbol{D}(\boldsymbol{x})\boldsymbol{x} - \boldsymbol{F}(\boldsymbol{x}) \\ \tilde{\boldsymbol{x}}(0) &= \boldsymbol{x}_0 \end{aligned} \tag{4.24}$$

is solved for $\tilde{\boldsymbol{x}}$. Define the operator $\mathcal{H}$ according to (4.24), i.e., $\tilde{\boldsymbol{x}} = \mathcal{H}\boldsymbol{x}$. Let $\tilde{\boldsymbol{y}} = \tilde{\mathcal{H}}(\boldsymbol{x})\boldsymbol{y}$ be the result of the following iteration applied to $\boldsymbol{y}$:

$$\begin{aligned} \left(\frac{d}{dt} + \boldsymbol{D}(\boldsymbol{x})\right)\tilde{\boldsymbol{y}} &= \boldsymbol{D}(\boldsymbol{x})\boldsymbol{y} - \boldsymbol{F}(\boldsymbol{y}) \\ \tilde{\boldsymbol{y}}(0) &= \boldsymbol{y}_0 \end{aligned} \tag{4.25}$$

Note that $\boldsymbol{D}$ is still evaluated at $\boldsymbol{x}$. Define $\boldsymbol{p} = \boldsymbol{y} - \boldsymbol{x}$ and $\tilde{\boldsymbol{p}} = \tilde{\boldsymbol{y}} - \tilde{\boldsymbol{x}}$. From (4.24) and (4.25), it is required that $\tilde{\boldsymbol{p}}$ satisfy

$$\begin{aligned} \left(\frac{d}{dt} + \boldsymbol{D}(\boldsymbol{x})\right)\tilde{\boldsymbol{p}} &= \boldsymbol{D}(\boldsymbol{x})\boldsymbol{p} - \boldsymbol{J}_F(\boldsymbol{x})\boldsymbol{p} + \mathcal{O}(\|\boldsymbol{p}\|^2) \\ \tilde{\boldsymbol{p}}(0) &= \boldsymbol{p}_0 \end{aligned}$$

or in operator equation formulation:

$$\tilde{\boldsymbol{p}} = \mathcal{K}\boldsymbol{p} = \tilde{\mathcal{H}}(\boldsymbol{x})\boldsymbol{y} - \mathcal{H}\boldsymbol{x} + \mathcal{O}(\|\boldsymbol{p}\|^2)$$

for $\boldsymbol{y} = \boldsymbol{x} + \boldsymbol{p}$.

In general, the operator $\mathcal{H}$ is defined by

$$(\mathcal{H}\boldsymbol{x})(t) = \boldsymbol{\Phi}_M(t,0)\tilde{\boldsymbol{x}}(0) + \int_0^t \boldsymbol{\Phi}_M(t,s)\left(\boldsymbol{M}(\boldsymbol{x}(s),s)\boldsymbol{x}(s) - \boldsymbol{F}(\boldsymbol{x}(s),s)\right) ds$$

or, equivalently, that $\tilde{\boldsymbol{x}} = \mathcal{H}\boldsymbol{x}$ satisfies

$$\begin{aligned} \left(\tfrac{d}{dt} + \boldsymbol{M}(\boldsymbol{x})\right)\tilde{\boldsymbol{x}} &= \boldsymbol{M}(\boldsymbol{x})\boldsymbol{x} - \boldsymbol{F}(\boldsymbol{x}) \\ \tilde{\boldsymbol{x}}(0) &= \boldsymbol{x}_0 \end{aligned}$$

The Jacobi-Newton iteration as described in (4.24) for $\boldsymbol{M} = \boldsymbol{D}$, computes $\mathcal{H}\boldsymbol{x}$ exactly, and then the operator-waveform product is approximately computed according to $\tilde{\mathcal{H}}(\boldsymbol{x})\boldsymbol{y} - \mathcal{H}\boldsymbol{x}$. Now, for the case $\boldsymbol{M} \neq \boldsymbol{D}$, a second approximation is introduced, i.e., the relaxation Newton for $\boldsymbol{M} \neq \boldsymbol{D}$ only *approximately* computes $\mathcal{H}(\boldsymbol{x})$.

Consider, as an example, taking one step of the Seidel-Newton waveform method for solving the nonlinear IVP. If the procedure is examined component-wise, the following sequence of scalar IVP's needs to be solved, where the dependence on t is omitted for clarity:

$$\begin{aligned} \tfrac{d}{dt}(\tilde{x}_1 - x_1) + \frac{\partial F_1(\boldsymbol{x})}{\partial x_1}(\tilde{x}_1 - x_1) &= -\tfrac{d}{dt}x_1 - F_1(\boldsymbol{x}) \\ \tfrac{d}{dt}(\tilde{x}_2 - x_2) + \frac{\partial F_2(\boldsymbol{x})}{\partial x_2}(\tilde{x}_2 - x_2) &= -\tfrac{d}{dt}x_2 - F_2(\tilde{x}_1, x_2, \ldots) \\ &\cdots \end{aligned}$$

The equation for $\tilde{x}_2$ can be approximated by expanding $F_2(\tilde{x}_1, x_2, \ldots)$ in a Taylor series about $\boldsymbol{x}$:

$$\begin{aligned} \tfrac{d}{dt}(\tilde{x}_2 - x_2) + \frac{\partial F_2(\boldsymbol{x})}{\partial x_2}(\tilde{x}_2 - x_2) &= -\tfrac{d}{dt}x_2 - F_2(\tilde{x}_1, x_2, \ldots) \\ &= -\tfrac{d}{dt}x_2 - F_2(\boldsymbol{x}) - \frac{\partial F_2(\boldsymbol{x})}{\partial x_1}(\tilde{x}_1 - x_1) + \mathcal{O}((\tilde{x}_1 - x_1)^2) \end{aligned}$$

so that

$$\tfrac{d}{dt}(\tilde{x}_2 - x_2) + \frac{\partial F_2(\boldsymbol{x})}{\partial x_1}(\tilde{x}_1 - x_1) + \frac{\partial F_2(\boldsymbol{x})}{\partial x_2}(\tilde{x}_2 - x_2) \approx -\tfrac{d}{dt}x_2 - F_2(\boldsymbol{x})$$

The Seidel-Newton waveform method can therefore be expressed compactly as approximately solving the system of equations:

$$\tfrac{d}{dt}\tilde{\boldsymbol{x}} + (\boldsymbol{L}(\boldsymbol{x}) + \boldsymbol{D}(\boldsymbol{x}))\tilde{\boldsymbol{x}} = (\boldsymbol{L}(\boldsymbol{x}) + \boldsymbol{D}(\boldsymbol{x}))\boldsymbol{x} - \boldsymbol{F}(\boldsymbol{x})$$

In this case, the approximation depends on $\|\tilde{\boldsymbol{x}} - \boldsymbol{x}\|^2$. In operator equation formulation, the Seidel-Newton waveform method calculates

$$\tilde{\boldsymbol{x}} = \mathcal{H}\boldsymbol{x} + \mathcal{O}(\|\mathcal{H}\boldsymbol{x} - \boldsymbol{x}\|^2)$$

The *WORDS* program computes $\mathcal{H}\boldsymbol{x}$ by incorporating the following small modifications to the main WR routine:

1. Use $\boldsymbol{x}^m(t+h)$ as the initial guess for $\boldsymbol{x}^{m+1}(t+h)$ within the Newton iteration in the numerical integration routine, instead of using an extrapolation based on values of $\boldsymbol{x}^{m+1}$ at previous time points.

2. Perform only one Newton iteration at each timestep of the integration method.

In order to calculate $\tilde{\mathcal{H}}(\boldsymbol{x})\boldsymbol{y}$, there is the additional requirement of linearizing $\boldsymbol{M}$ about $\boldsymbol{x}$ instead of $\boldsymbol{y}$. Since approximations are already involved in the level two operator waveform product, one additional approximation is made where $\mathcal{H}(\boldsymbol{x}+\boldsymbol{p})$ is used in place of $\tilde{\mathcal{H}}(\boldsymbol{x})(\boldsymbol{x}+\boldsymbol{p})$, introducing an error of $\mathcal{O}(\|\boldsymbol{p}\|)$. Then, $\tilde{\boldsymbol{p}}$ is calculated according to:

$$\tilde{\boldsymbol{p}} = \mathcal{K}\boldsymbol{p} = \mathcal{H}(\boldsymbol{x}+\boldsymbol{p}) - \mathcal{H}\boldsymbol{x} + \mathcal{O}(\|\mathcal{H}\boldsymbol{x}-\boldsymbol{x}\|^2) + \mathcal{O}(\|\boldsymbol{p}\|) + \mathcal{O}(\|\boldsymbol{p}\|^2).$$

Full Linearization Approach (Level Three)

The third, and most accurate approach, is to actually use the fully-linearized Jacobian. To implement this, the main WR routine in *WORDS* was augmented in the following manner:

1. Compute the full matrix $\boldsymbol{J}_F(\boldsymbol{x})$ — this requires additional data-structures as well as additional computation since the normal *WORDS* WR routine only requires the diagonal blocks of $\boldsymbol{J}_F(\boldsymbol{x})$.

2. Perform a matrix-vector product $\boldsymbol{J}_F(\boldsymbol{x})\boldsymbol{p}$ in place of the function evaluation for $\boldsymbol{F}$ — the result of the matrix-vector product is used as the right-hand side for the integration routine.

3. Perform only one Newton iteration at each timestep of the integration method (since the problem is linear).

This approach gives the exact operator-waveform product for the conjugate-direction methods, with the slight drawback of some extra function evaluations. Preliminary experiments showed that this extra work was more than offset by improved performance; thus the level three operator-waveform product was used for all experimental results reported in Section 4.5.

4.4.3 Inner Product

The inner product $\langle \boldsymbol{x}, \boldsymbol{y} \rangle$ on the space $\mathbb{H}$ is given by

$$\langle \boldsymbol{x}, \boldsymbol{y} \rangle = \sum_{i=1}^{N} \int_0^T x_i(t) y_i(t) dt.$$

A simple quadrature method (i.e., the trapezoidal rule) was used to perform the numerical integration.

4.4.4 Initial Residual

The initial residual is calculated in *WORDS* with the main WR routine as well. Interestingly, the initial residual can be approximately calculated by taking one step of a relaxation-Newton waveform method (in the present case, the Seidel-Newton waveform method).

Again consider computing the linearized operator-waveform product for a conjugate-direction method applied to the nonlinear IVP. The initial residual for the preconditioned linear IVP is simply $\boldsymbol{\psi} - (\boldsymbol{I} - \mathcal{K})\, \boldsymbol{x}^{m+1,0}$, or

$$\begin{aligned} \boldsymbol{r}^{m+1,0} &= \boldsymbol{\Phi}_M(t,0)\boldsymbol{x}^{m+1,0}(0) + \int_0^t \boldsymbol{\Phi}_M(t,s) \left(\boldsymbol{J}_F(\boldsymbol{x}^m(s),s)\boldsymbol{x}^m(s) - \boldsymbol{F}(\boldsymbol{x}^m(s),s) \right) ds \\ &\quad + \int_0^t \boldsymbol{\Phi}_M(t,s)\boldsymbol{N}(\boldsymbol{x}^m(s),s)\boldsymbol{x}^{m+1}(s) ds - \boldsymbol{x}^{m+1}(t) \\ &= \boldsymbol{\Phi}_M(t,0)\boldsymbol{x}_0 + \int_0^t \boldsymbol{\Phi}_M(t,s) \left(\boldsymbol{M}(\boldsymbol{x}^m(s),s)\boldsymbol{x}^m(s) - \boldsymbol{F}(\boldsymbol{x}^m(s),s) \right) ds - \boldsymbol{x}^{m+1}(t) \end{aligned}$$

Here, it is assumed that $\boldsymbol{x}^{m+1,0} = \boldsymbol{x}^m$.

Consider one step of the Jacobi-Newton waveform method, in which the IVP

$$\begin{aligned} \left(\tfrac{d}{dt} + \boldsymbol{D}(\boldsymbol{x}(t),t) \right) \tilde{\boldsymbol{x}} &= \boldsymbol{D}(\boldsymbol{x}(t),t)\boldsymbol{x} - \boldsymbol{F}(\boldsymbol{x}(t),t) \\ \tilde{\boldsymbol{x}}(0) = \boldsymbol{x}_0 & \end{aligned}$$

is solved for $\tilde{\boldsymbol{x}}$. In other words,

$$\tilde{\boldsymbol{x}}(t) = \boldsymbol{\Phi}_D(t,0)\tilde{\boldsymbol{x}}_0 + \int_0^t \boldsymbol{\Phi}_D(t,s) \left(\boldsymbol{D}(\boldsymbol{x}(s),s)\boldsymbol{x}(s) - \boldsymbol{F}(\boldsymbol{x}(s),s) \right) ds$$

which can be expressed in operator equation form as

$$\tilde{\boldsymbol{x}} = \mathcal{H}_D \boldsymbol{x}$$

Now, it should be easy to see that

$$\begin{aligned} \boldsymbol{r}^{m+1,0} &= \boldsymbol{\psi} - (\boldsymbol{I} - \mathcal{K}_D)\, \boldsymbol{x}^{m+1,0} \\ &= \boldsymbol{\psi} + \mathcal{K}_D \boldsymbol{x}^{m+1,0} - \boldsymbol{x}^{m+1,0} \\ &= \mathcal{H}_D \boldsymbol{x}^m - \boldsymbol{x}^m \end{aligned}$$

where $\mathcal{H}_D$ and $\mathcal{K}_D$ are defined by substituting $\boldsymbol{D}$ for $\boldsymbol{M}$ in the definitions of $\mathcal{H}$ and $\mathcal{K}$.

Here, it has been attempted to calculate $\mathcal{H}\boldsymbol{x}$ in the same way as for the level two operator-waveform product. That is, for $\boldsymbol{M} \neq \boldsymbol{D}$, one step of relaxation-Newton only approximately computes the $\mathcal{H}\boldsymbol{x}$. This is the only approximation involved for the residual calculation and is much less grievous than, say, the approximations made in the level one and level two operator-waveform product methods. In any case, *WORDS* uses the following relationship for the initial residual:

$$\boldsymbol{r}^{m+1,0} = \mathcal{H}\boldsymbol{x}^m - \boldsymbol{x}^m$$

4.5 Experimental Results

Four N-channel MOSFET examples were used to compare the performance of the relaxation and conjugate-direction waveform methods:

kG: 2.2 μm channel-length; 50 psec, 0-5V ramp on the gate with the drain at 5V.

kD: 2.2 μm channel-length; 50 psec, 0-5V ramp on the drain with the gate at 5V.

jG: 0.17 μm channel-length; 5 psec, 0-1V ramp on the gate with the drain at 1V.

jD: 0.17 μm channel-length; 5 psec, 0-1V ramp on the drain with the gate at 1V.

The parameters used with the conjugate-direction methods were: $\epsilon^0 = 0.1$, $\nu = \sqrt{0.1}$, and $\phi = 1\times10^{-18}$. To simplify comparisons, 32 equally-spaced timesteps were used in all experiments.

Table 4-1 shows the number of function evaluations and the CPU time required for each of the waveform methods to reduce the max-norm of the drain terminal current error below 0.01% of the max-norm of the drain terminal current. As Table 4-1 indicates, conjugate-direction methods significantly reduced the number of function evaluations and CPU time over WR and WRN. In fact, in the **jG** example, WCGS is 22 times faster than ordinary WR. As is common in the algebraic case, WGMRES and WGCR perform similarly in terms of function evaluations, but WGMRES is computationally more efficient because it avoids several waveform inner products on each iteration. As is also common in the algebraic case, WCGS performs very well on most problems, but can also exhibit convergence difficulty on others.

Note that the speedup in CPU time is not as impressive as the speedup in terms of function evaluations. This is partly due to the extra work required at each iteration of the conjugate-direction methods — however, careful profiling and hand optimization of

Example	Method	FEvals	CPU sec
jD	WR	8.43×10^6	14469
	WRN	3.77×10^6	7088
	WGCR	2.21×10^5	1138
	WGMRES	2.21×10^5	991
	WCGS	2.77×10^5	820
jG	WR	7.48×10^6	12615
	WRN	3.41×10^6	6214
	WGCR	1.97×10^5	1011
	WGMRES	1.97×10^5	877
	WCGS	1.97×10^5	568
kD	WR	1.22×10^6	1526
	WRN	3.94×10^5	559
	WGCR	9.03×10^4	315
	WGMRES	9.03×10^4	280
	WCGS	9.92×10^4	214
kG	WR	1.43×10^6	1756
	WRN	4.09×10^5	578
	WGCR	1.03×10^5	353
	WGMRES	1.03×10^5	316
	WCGS	Non-Convergence	

Table 4-1: Comparison of WR, WRN, WGCR, WGMRES, and WCGS. CPU times shown are for an IBM RS/6000 model 540.

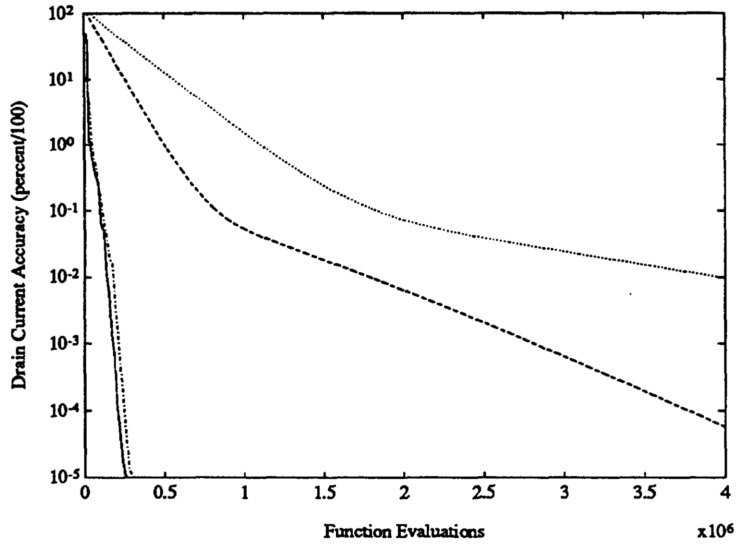

Figure 4-4: Convergence comparison between WR (dotted), WRN (dashed), WGCR/WGMRES (solid), and WCGS (dash-dotted) for **jD** example. The max-norm of the relative drain terminal current error is plotted against the number of function evaluations.

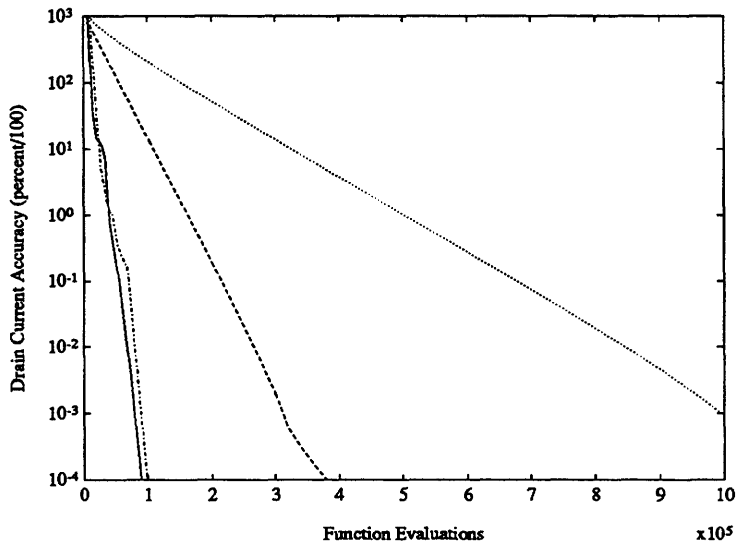

Figure 4-5: Convergence comparison between WR (dotted), WRN (dashed), WGCR/WGMRES (solid), and WCGS (dash-dotted) for **kD** example. The max-norm of the relative drain terminal current error is plotted against the number of function evaluations.

the conjugate-direction code will improve the CPU time comparison. The difference is especially apparent with WGMRES and WGCR, because the amount of extra work per iteration grows linearly with the number of iterations (the number of *linear* iterations per Newton iteration in the nonlinear versions). On the other hand, WCGS maintains the same amount of work per iteration but can suffer from occasional stability problems. Recently, Freund and Nachtigal have developed the QMR algorithm which is characterized by a minimization property over a Krylov space but which does not have a linearly increasing amount of work per iteration [26]. A function-space generalization of the QMR algorithm is a topic which is currently being investigated.

The graphs in Figures 4-4 and 4-5 compare the convergence of WR, WRN, WGCR, WGMRES, and WCGS for the **jD** and **kD** examples, respectively. In the graphs, the terminal current error versus number of function evaluations is plotted and clearly demonstrates the rapid convergence of the conjugate-direction methods.

4.6 Conclusion

In this chapter, some new dynamic iterative methods were presented to accelerate the convergence of the WR algorithm. The methods are based on the application of the Galerkin method to an operator equation formulation of the linear time-varying initial-value problem. Experimental results demonstrated that this acceleration significantly reduces the computation time for device transient simulation.

Future work is primarily focused on developing theoretical results about the convergence of linear and nonlinear conjugate-direction methods for differential-algebraic systems of equations. Expectations are to be able to provide intuitive convergence rates as in [3] and [27]. Other open questions remain regarding the behavior of the conjugate direction methods in the presence of multirate integration methods as well as in the presence of errors due to numerical quadrature. Finally, the function-space generalization of the QMR algorithm is under further study.

References

[1] W. Engl, R. Laur, and H. Dirks, "MEDUSA – A simulator for modular circuits," *IEEE Trans. CAD*, vol. 1, pp. 85–93, April 1982.

[2] M. Reichelt, J. White, and J. Allen, "Waveform relaxation for transient two-dimensional simulation of MOS devices," in *International Conference on Computer Aided-Design*, (Santa Clara, California), pp. 412–415, November 1989.

[3] U. Miekkala and O. Nevanlinna, "Convergence of dynamic iteration methods for initial value problems," *SIAM J. Sci. Stat. Comp.*, vol. 8, pp. 459–467, 1987.

[4] R. W. Brockett, *Finite Dimensional Linear Systems.* New York: Wiley, 1970.

[5] J. B. Conway, *A Course in Functional Analysis, Second Edition.* New York: Springer-Verlag, 1990.

[6] R. Kress, *Linear Integral Equations.* New York: Springer-Verlag, 1989.

[7] K. E. Atkinson, *A Survey of Numerical Methods for the Solution of Fredholm Integral Equations of the Second Kind.* Philadelphia: SIAM, 1976.

[8] P. Linz, *Analytical and Numerical Methods for Volterra Equations.* Philadelphia: SIAM, 1985.

[9] J. K. White and A. Sangiovanni-Vincentelli, *Relaxation Techniques for the Simulation of VLSI Circuits.* Engineering and Computer Science Series, Norwell, Massachusetts: Kluwer Academic Publishers, 1986.

[10] R. C. MacCamy and P. Weiss, "Numerical solution of Volterra integral equations," *Nonlinear Anal.*, vol. 3, pp. 677–695, 1979.

[11] G. Miel, "Iterative refinement of the method of moments," *Numer. Funct. Anal. and Optimiz.*, vol. 9(11-12), pp. 1193–1200, 1987–1988.

[12] P. Omari, "On the fast convergence of a Galerkin-like method for equations of the second kind," *Math. Z.*, vol. 201, pp. 529–539, 1989.

[13] Y. V. Vorobyev, *Method of Moments in Applied Mathematics.* New York: Gordon and Breach, 1965.

[14] E. Zeidler, *Nonlinear Functional Analysis and its Applications.* New York: Springer-Verlag, 1985.

[15] S. G. Mikhlin, *Variational Methods in Mathematical Physics.* New York: Macmillan, 1964.

[16] Y. Saad and M. Schultz, "GMRES: A generalized minimum residual algorithm for solving nonsymmetric linear systems," *SIAM J. Sci. Statist. Comput.*, vol. 7, pp. 856–869, July 1986.

[17] H. C. Elman, *Iterative Methods for Large Sparse Nonsymmetric Systems of Linear Equations.* PhD thesis, Computer Science Dept., Yale University, New Haven, CT, 1982.

[18] R. Saleh and J. White, "Accelerating relaxation algorithms for circuit simulation using waveform-newton and step-size refinement," *IEEE Trans. CAD*, vol. 9, no. 9, pp. 951–958, 1990.

[19] P. Brown and Y. Saad, "Hybrid Krylov methods for nonlinear systems of equations," *SIAM J. Sci. Statist. Comput.*, vol. 11, pp. 450–481, May 1990.

[20] R. Bank, W. Coughran, Jr., W. Fichtner, E. Grosse, D. Rose, and R. Smith, "Transient simulation of silicon devices and circuits," *IEEE Trans. CAD*, vol. 4, pp. 436–451, October 1985.

[21] S. Selberherr, *Analysis and Simulation of Semiconductor Devices.* New York: Springer-Verlag, 1984.

[22] D. Scharfetter and H. Gummel, "Large-signal analysis of a silicon read diode oscillator," *IEEE Transactions on Electron Devices*, vol. ED-16, pp. 64–77, January 1969.

[23] K. Mayaram and D. Pederson, "CODECS: A mixed-level device and circuit simulator," in *International Conference on Computer Aided-Design*, (Santa Clara, California), pp. 112–115, November 1988.

[24] E. Lelarasmee, A. E. Ruehli, and A. L. Sangiovanni-Vincentelli, "The waveform relaxation method for time domain analysis of large scale integrated circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 1, pp. 131–145, July 1982.

[25] P. Sonneveld, "CGS, a fast Lanczos-type solver for nonsymmetric linear systems," *SIAM J. Sci. Statist. Comput.*, vol. 10, pp. 36–52, 1989.

[26] R. W. Freund and N. M. Nachtigal, "QMR: A quasi-minimal residual method for non-Hermitian linear systems," Tech. Rep. 90.51, RIACS, NASA Ames Research Center, December 1990.

[27] U. Miekkala, "Dynamic iteration methods applied to linear DAE systems," *J. Comput. Appl. Math.*, vol. 25, pp. 133–151, 1989.

5 Conclusions

The observation was made in the introduction that for specific initial value problems, one can achieve the largest performance gains by using closely matched algorithms and architectures to exploit characteristic features of the particular problem to be solved. This claim was well borne out by the results obtained by both the circuit and device simulation problems.

In developing algorithms for the vision circuits in Chapter 3, the fact was exploited that the circuits have a regular structure which maps nicely to a massively parallel architecture. Moreover, the coupling between cells in these arrays is such that block iterative methods could be used to solve the equations generated by an implicit time-discretization scheme. The experimental results were very encouraging. A circuit having over 300,000 SPICE level 3 MOS devices was simulated in about half an hour, compared to three days using even the best serial algorithm running on a Sun4/490. The simulation capability provided by CMVSIM should make it a useful tool in the design cycle of real robotic vision circuits.

In Chapter 4, a new class of waveform methods was developed to accelerate the convergence of the WR algorithm. Although the conjugate direction waveform methods are more general than the techniques developed for the vision circuits, they were particularly effective in reducing the computation time for performing device transient simulation. In the best cases, computation time was reduced by more than an order of magnitude over the conventional WR algorithm.