Accelerated Waveform Methods for Parallel Transient Simulation of Semiconductor Devices

Andrew Lumsdaine, *Member, IEEE*, Mark W. Reichelt, *Member, IEEE*, Jeffrey M. Squyres, and Jacob K. White, *Member, IEEE*

***Abstract*— Simulating transients in semiconductor devices involves numerically solving the time-dependent drift-diffusion equations, usually in two or three space dimensions. Because of the computation cost of these simulations, methods that perform careful domain decomposition so as to exploit parallel processing have received much recent attention. In this paper, we describe using accelerated waveform relaxation (WR) to perform parallel device transient simulation using both clusters of workstations and the IBM SP-2. The accelerated WR algorithms are compared to pointwise direct and iterative methods, and it is shown that the accelerated WR method is competitive on a single processor. In addition, it is shown that with a domain decomposition chosen for rapid iterative method convergence rather than parallel efficiency, the pointwise methods parallelize poorly but the WR method achieves near linear speedup (with respect to the number of processors) on the IBM SP-2.**

I. Introduction

THE GROWING importance and computational expense of performing semiconductor device transient simulation and the increasing availability of parallel computers suggest that parallel algorithms be developed and used for this problem. Results in [1] and [2] demonstrate that SIMD type parallel machines can be used effectively for device transient simulation. However, special-purpose SIMD machines have not been cost-effective enough (in terms of hardware or software) to gain wide-spread use and it seems that instead, the near-term future of parallel computing will be dominated by medium-grain MIMD machines. One striking feature of many modern MIMD machines is that the computing power in the nodes is typically provided by workstation level microprocessors. Indeed, workstation clusters are themselves a viable parallel computing resource and with the advent of machines such as the IBM SP-1 and SP-2, the distinction between workstation clusters and "real" parallel machines becomes increasingly blurred. Because the computing nodes of loosely coupled, workstation-based parallel computers are themselves general purpose computers, the advantages of these parallel machines are low-cost and high general utility. The disadvantages of these machines, particularly of workstation clusters, are high communication latency and limited communication bandwidth.

To obtain high parallel performance on a loosely coupled MIMD parallel computer, it is critical that a numerical method avoid frequent parallel synchronization [3]. For the application of semiconductor device transient simulation, this has been achieved by careful domain partitioning, for example see [4]. The waveform relaxation (WR) approach to solving time-dependent initial-value problems reduces parallel synchonization using an almost orthogonal approach to careful domain partitioning. The iterates are vector waveforms over an interval, rather than vectors at single timepoints [5]–[7]. As with any iterative scheme, overall computational efficiency of WR depends on rapid convergence, and there have been several investigations into accelerating WR [5], [8], including using multigrid [9], [10], Krylov-subspace [11], [12], and convolution successive overrelaxation (CSOR) techniques [13].

In this paper, we extend the results in [11]–[14] and provide experimental results using waveform methods on two different parallel machines (a cluster of SPARC workstations and an IBM SP-2) for performing transient simulation on a variety of MOS devices. The experimental results show that parallel waveform methods are far less sensitive to domain partitioning than parallelized versions of the standard serial approach for performing device transient simulation. The strong implication of the results is that, as MIMD machines (especially workstation clusters) become more prevalent, waveform methods will gain in importance for all areas of computational science and engineering that require the solution of time-dependent problems.

We want to emphasize that we are not attempting to develop *the best* parallel device simulator in this paper. Rather, we wish to carefully study a number of waveform and pointwise methods using the same device simulator and domain partitioning, so as to understand the effects of different parallel environments on pointwise versus waveform methods. Thus, to the greatest extent possible, we are attempting to compare only these numerical algorithms (keeping all else about the device simulation process constant) using a simulation program that incorporates most, but not all, of the important semiconductor device physics. So, the waveform methodologies presented here are general strategies for increasing the parallel performance of time-dependent problems. In this paper, we show the experimental results for accelerated waveform methods

Manuscript received March 7, 1995; revised June 20, 1995, August 18, 1995, and December 19, 1995. This work was supported by a grant from IBM, by the Defense Advanced Research Projects Agency under Contract N00014-91-J-1698, and by the National Science Foundation under Contracts MIP-8858764 A02 and CCR92-09815. This paper was recommended by Associate Editor S. Duvall.

A. Lumsdaine and J. M. Squyres are with the Laboratory for Scientific Computing, Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN 46556 USA.

M. W. Reichelt is with The MathWorks Inc., Natick, MA 01760 USA.

J. K. White is with the Research Laboratory of Electronics, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139 USA.

Publisher Item Identifier S 0278-0070(96)05041-5.

applied to a particular device simulator using particular discretization and having particular physical models. However, the waveform approaches themselves are independent of these and the approaches presented here would certainly be applicable to, e.g., simulators with more sophisticated physics and with unstructured meshes.

We begin in Section II by reviewing the transient simulation of semiconductor devices. In Section III, we review ordinary WR and the standard serial methods used to solve the device equations. We then discuss two of the most effective acceleration techniques for waveform methods, namely CSOR and Krylov-subspace acceleration. Finally, in Section IV, we apply the methods to device simulation on both serial and parallel machines, and give conclusions and acknowledgments in Sections VI and V.

II. Device Transient Simulation

Charge transport within a semiconductor device is assumed to be governed by the Poisson equation, and the electron and hole continuity equations

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

where u is the normalized electrostatic potential in thermal volts, n and p are the electron and hole concentrations, $\boldsymbol{J}_n$ and $\boldsymbol{J}_p$ are the electron and hole current densities, N_D and N_A are the donor and acceptor concentrations, R is the net generation and recombination rate, q is the magnitude of electronic charge, and ϵ is the spatially dependent dielectric permittivity [15], [16].

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

$$\begin{aligned}\boldsymbol{J}_n&=-q\mu_n n\,\nabla\left(\frac{kT}{q}\,u\right)+qD_n\nabla n\\&=\quad -kT\mu_n n\nabla u+qD_n\nabla n\\\boldsymbol{J}_p&=-q\mu_p p\,\nabla\left(\frac{kT}{q}\,u\right)-qD_p\nabla p\\&=\quad -kT\mu_p p\nabla u-qD_p\nabla p\end{aligned}$$

where μ_n and μ_p are the electron and hole mobilities, and D_n and D_p are the diffusion coefficients. The mobilities μ_n and μ_p may be computed as nonlinear functions of the electric field E, e.g.

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

where v_{sat} and β are constants and μ_{n_0} is a doping-dependent mobility [17]. The diffusion constants D_n and D_p are related to the mobilities by the Einstein relations

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

Using the Scharfetter–Gummel method [18] to spatially discretize an N-node rectangular mesh covering a two-dimensional slice of a MOSFET yields a sparsely coupled differential-algebraic initial value problem (IVP) consisting of $3N$ equations in $3N$ unknowns, denoted by

$$\begin{array}{rlrl}\boldsymbol{F}_1(\boldsymbol{u}(t),\boldsymbol{n}(t),\boldsymbol{p}(t))&=0&\boldsymbol{u}(0)&=\boldsymbol{u}_0\\\frac{d}{dt}\boldsymbol{n}(t)+\boldsymbol{F}_2(\boldsymbol{u}(t),\boldsymbol{n}(t),\boldsymbol{p}(t))&=0\ \text{and}&\boldsymbol{n}(0)&=\boldsymbol{n}_0\\\frac{d}{dt}\boldsymbol{p}(t)+\boldsymbol{F}_3(\boldsymbol{u}(t),\boldsymbol{n}(t),\boldsymbol{p}(t))&=0&\boldsymbol{p}(0)&=\boldsymbol{p}_0\end{array}\qquad(1)$$

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

$$F_{1_i}(u_i,n_i,p_i,u_j)=\frac{kT}{q}\sum_j\left\{\frac{d_{ij}\epsilon_{ij}}{L_{ij}}\Big[u_i-u_j\Big]\right\}-qA_i(p_i-n_i+N_{D_i}-N_{A_i})\qquad(2)$$

$$F_{2_i}(u_i,n_i,u_j,n_j)=\frac{kT}{qA_i}\sum_j\left\{\frac{d_{ij}\mu_{n_{ij}}}{L_{ij}}\Big[n_iB(u_i-u_j)-n_jB(u_j-u_i)\Big]\right\}+R_i\qquad(3)$$

$$F_{3_i}(u_i,p_i,u_j,p_j)=\frac{kT}{qA_i}\sum_j\left\{\frac{d_{ij}\mu_{p_{ij}}}{L_{ij}}\Big[p_iB(u_j-u_i)-p_jB(u_i-u_j)\Big]\right\}+R_i.\qquad(4)$$

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

III. Accelerated Waveform Methods

The standard approach used to solve the differential-algebraic equation (DAE) system (1) is to discretize the system in time with a low-order implicit integration method such as the second-order backward difference formula. For an N-node mesh, the resulting sequence of nonlinear algebraic systems in $3N$ unknowns is typically solved with some variant of Newton's method and/or relaxation [15], [19]. 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 (1) with a parallel computer is to decompose the DAE system into subsystems before time discretization. The system is then solved iteratively by solving the subsystems independently, using fixed waveforms from previous iterations for the variables from other subsystems. This dynamic iteration process, applying relaxation directly to the DAE system, is known as WR [7] or as the Picard–Lindelöf iteration [5]. Applying WR to the device simulation problem yields Algorithm 3.1 [14] at the bottom of the page.

The WR algorithm has several computational advantages. Since it is an iterative method, WR avoids factoring large sparse matrices. WR can exploit multirate behavior, using different timesteps to resolve different solution components. Finally, WR is well suited to parallel computation, because of a low communication/computation ratio. However, when applied to solving the device simulation equation system (1), the WR algorithm converges slowly, unless acceleration techniques are applied [11], [14]. In the following two sections, we review two of the most effective acceleration techniques for WR, namely CSOR [13] and Krylov-subspace acceleration [11].

CSOR

CSOR is a waveform extension of the standard linear algebraic successive overrelaxation (SOR) technique, and has been shown to dramatically accelerate the convergence rate of WR for devices [13]. The CSOR algorithm is most readily described in the context of numerically solving the linear time-invariant initial-value problem

$$\left(\frac{d}{dt} + \boldsymbol{A}\right)\boldsymbol{x}(t) = \boldsymbol{b}(t) \quad \text{with} \quad \boldsymbol{x}(0) = \boldsymbol{x}_0 \tag{5}$$

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

A WR algorithm using CSOR for solving (6) is shown in Algorithm 3.2. In iteration $k+1$, each waveform $\hat{x}_i^{k+1}$ is computed as in ordinary Gauss–Seidel WR, and then is moved slightly farther in the iteration direction by convolution with a CSOR function $\omega(t)$. The motivation for this approach is that in the frequency domain the initial-value problem is a linear system which can be solved with SOR having an optimal over-relaxation parameter at each frequency. This frequency-dependent SOR in the frequency domain translates to convolution in the time domain.

Algorithm 3.2 (Gauss–Seidel WR with CSOR Acceleration)

1) *Initialize:* Pick vector waveform $\boldsymbol{x}^0(t) \in (\mathbb{R}^n, [0,T])$ with $\boldsymbol{x}^0(0) = \boldsymbol{x}_0$.
2) *Iterate:* For $k = 0, 1, \cdots$ until converged
   - *Solve:* For scalar waveform $\hat{x}_i^{k+1}(t) \in (\mathbb{R}, [0,T])$ with $\hat{x}_i^{k+1}(0) = x_{0_i}$:

$$\left(\frac{d}{dt} + a_{ii}\right)\hat{x}_i^{k+1}(t) = b_i(t) - \sum_{j=1}^{i-1} a_{ij}x_i^{k+1}(t) - \sum_{j=i+1}^{n} a_{ij}x_i^{k}(t).$$

   - *Overrelax:* To generate $x_i^{k+1}(t) \in (\mathbb{R}, [0,T])$:

$$x_i^{k+1}(t) \leftarrow x_i^k(t) + \int_0^t \omega(\tau)\cdot\left[\hat{x}_i^{k+1}(t-\tau) - x_i^k(t-\tau)\right]d\tau. \tag{6}$$

In a practical implementation, the CSOR method is used to solve a problem that has been discretized in time with a multistep integration method. The overrelaxation convolution integral (8) is replaced with a convolution sum

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

Like the standard algebraic SOR method, the practical difficulty is in determining an appropriate overrelaxation parameter, in this case, sequence $\omega[m]$.

To determine the CSOR sequence $\omega[m]$, consider the linear IVP (6) discretized in time with the backward Euler method and globally uniform timesteps h. This reduces the linear IVP to a sequence of algebraic systems, one at each time point

$$\frac{\boldsymbol{x}[m] - \boldsymbol{x}[m-1]}{h} + \boldsymbol{A}\,\boldsymbol{x}[m] = \boldsymbol{b}[m] \quad \text{with} \quad \boldsymbol{x}[0] = \boldsymbol{x}_0 \tag{7}$$

where $\boldsymbol{x}[m]$ denotes the discrete approximation to $\boldsymbol{x}(t)$ at $t = mh$. Taking the unilateral z-transform of both sides of (10) yields

$$\left(\frac{1-z^{-1}}{h}\boldsymbol{I} + \boldsymbol{A}\right)\boldsymbol{x}(z) = \boldsymbol{b}(z)$$

where $z \in \mathbb{C}$ [20]. Using the relaxation splitting $\boldsymbol{A} = \boldsymbol{D} - \boldsymbol{L} - \boldsymbol{U}$, where $\boldsymbol{D}$, $\boldsymbol{L}$ and $\boldsymbol{U}$ are the diagonal, strictly lower triangular and strictly upper triangular pieces of $\boldsymbol{A}$, the Gauss–Jacobi WR iteration equation may be written as $\Delta\boldsymbol{x}^{k+1}(z) = \boldsymbol{H}_{GJ}(z)\ \Delta\boldsymbol{x}^k(z)$, where

$$\boldsymbol{H}_{GJ}(z) = \left(\frac{1-z^{-1}}{h}\boldsymbol{I} + \boldsymbol{D}\right)^{-1}(\boldsymbol{L} + \boldsymbol{U})$$

and $\Delta\boldsymbol{x}^{k+1}(z) = \boldsymbol{x}^{k+1}(z) - \boldsymbol{x}^k(z)$. Clearly, the spectrum of $\boldsymbol{H}_{GJ}(z)$ depends on z.

Algorithm 3.1 (Gauss–Jacobi WR for Device Simulation)

1) *Initialize:* Pick u^0, n^0, p^0 waveforms at all nodes.
2) *Iterate:* For $k = 0, 1, \cdots$ until converged
   - *Iterate:* For each node i, solve for $u_i^{k+1}, n_i^{k+1}, p_i^{k+1}$ waveforms:

$$\begin{array}{rclcl} & & F_{1_i}(u_i^{k+1}, n_i^{k+1}, p_i^{k+1}, u_j^k) = 0 & & u_i^{k+1}(0) = u_{0_i} \\ \frac{d}{dt}n_i^{k+1}(t) & + & F_{2_i}(u_i^{k+1}, n_i^{k+1}, u_j^k, n_j^k) = 0 & \text{with} & n_i^{k+1}(0) = n_{0_i} \\ \frac{d}{dt}p_i^{k+1}(t) & + & F_{3_i}(u_i^{k+1}, p_i^{k+1}, u_j^k, p_j^k) = 0 & & p_i^{k+1}(0) = p_{0_i} \end{array}$$

In a result reminiscent of classical SOR theory [21], [22], it can be shown that for a common, but restricted, class of matrices $\boldsymbol{A}$, the z-transform $\omega_{opt}(z)$ of the optimal CSOR sequence $\omega_{opt}[m]$ for solving the linear IVP (10) with Algorithm 3.2 may be expressed as

$$\omega_{opt}(z) = \frac{2}{1+\sqrt{1-\mu_1(z)^2}} \tag{8}$$

where $\mu_1(z)$ is the largest-magnitude eigenvalue of the Gauss–Jacobi WR operator $\boldsymbol{H}_{GJ}(z)$ [13]. Fortunately, the CSOR method inherits some of the robustness of the algebraic SOR method, and the optimal parameter formula (11) can be successfully applied to a wider class of problems such as (1). Like implementations of the algebraic SOR method, the primary practical difficulty of computing $\omega_{opt}[m]$ is in computing a close approximation of the Gauss–Jacobi WR largest-magnitude eigenvalue $\mu_1(z)$. This problem is made more difficult by the eigenvalue's z-dependence.

One successful approach is to compute $\omega_{opt}[m]$ before beginning any WR iterations. First, power iterations are used to estimate the largest-magnitude eigenvalue $\mu_1(z)$ at several specific values of z (e.g., $z = 1, -1, \infty, \cdots$). Note that for real values of z, this simply amounts to adding $(1 - z^{-1})/h$ to the diagonal elements of $\boldsymbol{A}$ and computing the algebraic Gauss–Jacobi spectral radius. These values of $\mu_1(z)$ are used in formula (11) to compute values of $\omega_{opt}(z)$. Next, the computed values of $\omega_{opt}(z)$ are fitted by a ratio of low-order polynomials in z^{-1}

$$\omega_{opt}(z) \approx \frac{\sum_{k=0}^{M} b_k z^{-k}}{\sum_{k=0}^{N} a_k z^{-k}} = \sum_{k=0}^{N} \frac{c_k}{1 - r_k z^{-1}}. \tag{9}$$

Finally, the inverse z-transform is applied, yielding

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

Note that because the resulting $\omega_{opt}[m]$ is a simple sum of exponentials in time, the computational expense of the overrelaxation convolution is reduced to that of only a few multiplications and accumulations at each time point. A summary of the computation of $\omega_{opt}[m]$ is given in Algorithm 3.3.

Algorithm 3.3 (Approximation of $\omega_{opt}[m]$)

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

Of course, because the device simulation problem is nonlinear, the CSOR theory does not directly apply. And even when the time-discretized problem is linearized about some point, the linearization is time-dependent, and the resulting matrices, though block consistently ordered, do not necessarily satisfy the conditions of the optimal CSOR parameter theory. Nevertheless, the CSOR method is sufficiently robust in practice so that it can successfully be applied to the device problem. To obtain the results of Section V, the "optimal" CSOR parameter was determined by linearizing the device problem about the solution at time zero, and fitting $\omega_{opt}(z)$ with a rational function as described above. Also, to diminish the effect of the nonlinearity, the overrelaxation convolution was applied only to the potential variables $\boldsymbol{u}$.

Krylov-Subspace Techniques

A compact description of WR for solving (6) is shown in Algorithm 3.4 (with the splitting $\boldsymbol{A} = \boldsymbol{M} - \boldsymbol{N}$).

Algorithm 3.4 (WR for Linear Systems)

1) *Initialize:* Pick $\boldsymbol{x}^0$
2) *Iterate:* For $k = 0, 1, \cdots$ until converged

$$\text{Solve}\ \begin{array}{rcl} (\frac{d}{dt} + \boldsymbol{M})\boldsymbol{x}^{k+1}(t) & = & \boldsymbol{N}\boldsymbol{x}^k(t) + \boldsymbol{b}(t) \\ \boldsymbol{x}^{k+1}(0) & = & \boldsymbol{x}_0 \end{array}$$

for $\boldsymbol{x}^{k+1}$ on $[0, T]$.

The solution $\boldsymbol{x}$ to (6) is thus a fixed point of the WR algorithm, satisfying the integral operator equation

$$(\boldsymbol{I} - \mathcal{K})\boldsymbol{x} = \boldsymbol{\psi}. \tag{10}$$

Here, we define (13) on the function space $\mathbb{H} = \mathbb{L}_2([0,T], \mathbb{R}^N)$, $\boldsymbol{I} : \mathbb{H} \to \mathbb{H}$ is the identity operator, $\mathcal{K} : \mathbb{H} \to \mathbb{H}$ is defined by

$$(\mathcal{K}\boldsymbol{x})(t) = \int_0^t e^{(s-t)\boldsymbol{M}} \boldsymbol{N}\boldsymbol{x}(s)ds$$

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

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

WR for solving (13) is expressed in operator equation form simply as

$$\boldsymbol{x}^{k+1} = \mathcal{K}\boldsymbol{x}^k + \boldsymbol{\psi}. \tag{11}$$

Since for any finite interval $[0, T]$, the operator $\mathcal{K}$ is a Volterra integral operator (and thus has zero spectral radius [23]), this process will ultimately produce iterates $\boldsymbol{x}^k$ that converge to the solution $\boldsymbol{x}$ of (13), or equivalently, to the solution of (6). A more detailed analysis of convergence can be derived by considering (13) on the interval $[0, \infty)$ in which case $\mathcal{K}$ has nonzero spectral radius [5].

As shown in [12], Krylov-subspace methods can be applied to (13) to accelerate the convergence of WR, but as $\mathcal{K}$ is not self-adjoint, a variant suitable for nonself-adjoint operators must be used. One such method is waveform GMRES (WGM-

RES), an extension of the generalized minimum residual algorithm (GMRES) [24] to the space $\mathbb{H}$ and is shown in Algorithm 3.5.

Algorithm 3.5 (Waveform GMRES)

```
1) Start: Set $\boldsymbol{r}^0 = \boldsymbol{\psi} - (\boldsymbol{I} - \mathcal{K})\boldsymbol{x}^0$, $\boldsymbol{v}^1 = \boldsymbol{r}^0/\|\boldsymbol{r}^0\|$, $\beta = \|\boldsymbol{r}^0\|$
2) Iterate: For $k = 1, 2, \cdots$, until converged:
    • $h_{j,k} = \langle(\boldsymbol{I} - \mathcal{K})\boldsymbol{v}^k, \boldsymbol{v}^j\rangle$, $j = 1, 2, \cdots, k$
    • $\hat{\boldsymbol{v}}^{k+1} = (\boldsymbol{I} - \mathcal{K})\boldsymbol{v}^k - \sum_{j=1}^{k} h_{j,k}\boldsymbol{v}^j$
    • $h_{k+1,k} = \|\hat{\boldsymbol{v}}^{k+1}\|$
    • $\boldsymbol{v}^{k+1} = \hat{\boldsymbol{v}}^{k+1}/h_{k+1,k}$
3) Form approximate solution:
    • $\boldsymbol{x}^k = \boldsymbol{x}^0 + \boldsymbol{V}^k\boldsymbol{y}^k$, where $\boldsymbol{y}^k$ minimizes $\|\beta e_1 - \bar{\boldsymbol{H}}^k\boldsymbol{y}^k\|$
```

The two fundamental operations in Algorithm 3.5 are the operator-function product, $(\boldsymbol{I} - \mathcal{K})\boldsymbol{p}$, and the inner product, $\langle\cdot,\cdot\rangle$. When solving (13) in the space $\mathbb{H}$, these operations are as follows:

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

1) Solve $(\frac{d}{dt} + \boldsymbol{M})\boldsymbol{y} = \boldsymbol{N}\boldsymbol{p}$ with $\boldsymbol{y}(0) = \boldsymbol{p}_0 = 0$ for $\boldsymbol{y}(t)$, $t \in [0, T]$ to obtain $\boldsymbol{y} = \mathcal{K}\boldsymbol{p}$.
2) Set $\boldsymbol{w} = \boldsymbol{p} - \boldsymbol{y}$

Inner Product: The inner product $\langle\boldsymbol{x}, \boldsymbol{y}\rangle$ is given by $\langle\boldsymbol{x}, \boldsymbol{y}\rangle = \sum_{i=1}^{N}\int_0^T x_i(t)y_i(t)dt$.

Step 1 of the operator-function product is equivalent to one step of WR, hence WGMRES has as its core the standard WR iteration (with the concomitant parallelizability). Moreover, the inner product can be computed 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.

Another Krylov-subspace method that is suitable for accelerating WR is the conjugate gradient squared algorithm (CGS) [25]. The waveform CGS (WCGS) algorithm is a straightforward extension of CGS in which, like WGMRES, the matrix-vector products are replaced by operator-waveform products, and the vector inner products are replaced by waveform inner products. Because this extension is so straightforward, the WCGS algorithm will not be listed here, but a description of WCGS can be found in [26].

To use the waveform Krylov-subspace methods on the nonlinear device system (1), Newton's method is applied to (1), in a process sometimes referred to as the waveform Newton method (WN) [27], to obtain the following iteration

$$\left(\frac{d}{dt} + \boldsymbol{J}_F(\boldsymbol{x}^m)\right)\boldsymbol{x}^{m+1} = \boldsymbol{J}_F(\boldsymbol{x}^m)\boldsymbol{x}^m - \boldsymbol{F}(\boldsymbol{x}^m) \quad \text{with} \quad \boldsymbol{x}^{m+1}(0) = \boldsymbol{x}_0. \tag{12}$$

Here, $\boldsymbol{J}_F$ is the Jacobian of $\boldsymbol{F}$. We note that (15) is a linear time-varying system to be solved for $\boldsymbol{x}^{m+1}$, which can be accomplished with WGMRES [which can trivially extended to the time-varying version of (6)]. The resulting WN/WGMRES algorithm, a member of the class of hybrid Krylov methods [28], is given in Algorithm 3.6.

Algorithm 3.6 (Waveform Newton/WGMRES)

```
1) Initialize: Pick $\boldsymbol{x}^0$
2) Iterate: For $m = 0, 1, \cdots$ until converged
    • Linearize (1) to form (15)
    • Solve (15) with WGMRES
    • Update $\boldsymbol{x}^{m+1}$
```

IV. Parallel Implementation

The following solution methods were incorporated into the WR-based device transient simulation program pWORDS.

- Pointwise with direct methods (sparse Gaussian elimination) used to solve linear systems at each timestep (this method was not parallelized);
- Pointwise with preconditioned conjugate gradient squared (CGS) used to solve linear systems at each timestep;
- Waveform Relaxation (WR);
- Waveform Relaxation Newton (WRN);
- Convolution SOR Newton (CSORN);
- Waveform Newton/Waveform GMRES (WN/WGMRES);
- Waveform Newton/Waveform CGS (WN/WCGS).

The program was written in C using a message-passing single-program multiple-data (SPMD) paradigm. The message-passing was effected using the MPICH implementation [29] (from Argonne National Labs and Mississippi State University) of the Message Passing Interface (MPI) standard [30]. Because the code was written in C and because implementations of MPI exist for many different computing platforms, the same source code can be compiled for a large variety of environments. The same source code was used to produce both the cluster-based results and the SP-2 results reported here.

Partitioning

In this section, we describe the partitioning and communication patterns used for the parallel waveform solution methods and for the iterative pointwise solution methods.

As reported in [14], the node-by-node Gauss–Jacobi WR algorithm will typically require many hundreds (or even thousands) of iterations to converge, severely limiting the efficiency of WR-based device simulation. Instead of using a node-by-node approach, it is more effective to group nodes together into blocks and to solve for the corresponding variables simultaneously. Besides improving convergence of iterative methods, grouping nodes together provides an appropriate granularity for parallel processing on medium-grain MIMD processors. The pWORDS program supports arbitrary blocking schemes, but one blocking strategy that has been shown to be particularly effective in accelerating the convergence of iterative methods in MOSFET simulation [14] is to group the nodes in each vertical line of the discretization mesh together. Note, though such an approach best accelerates convergence, it is not a particularly effective blocking scheme for use on a parallel processor [4]. Alternative coarse blocking schemes would be suitable for applications using irregular discretization meshes.

In the SPMD model of parallel computation (used by pWORDS), identical programs are executed on all nodes (one of which is designated to be responsible for file and terminal I/O). To begin a parallel WR computation, the designated I/O node program reads in the device input file that specifies the device geometry and discretization mesh as well as the voltage boundary conditions imposed on the device. This information is then broadcast to the other nodes. The designated I/O program then partitions the device mesh into vertical-line blocks and assigns blocks to each node (including itself). The blocks are assigned to compute nodes so that all communication between compute nodes is between nearest neighbors. In addition to the vertical-line blocks assigned to it, each compute node also contains storage for the two vertical lines on either side of the contiguous block. These "pseudolines" are used only to store the solutions generated and communicated by the compute nodes solving those adjacent vertical lines.

Processing

The pWORDS program offers two different types of processing for the vertical-line blocks: Jacobi and Seidel. In our experience, Seidel order processing (which is required at any rate by the CSOR algorithm) always provided better performance (in terms of computation time) so we omit discussion of Jacobi order processing. For maximum parallelism, the Seidel approach uses red/black ordering which produces a block consistently ordered problem [21]. Since all of the waveform methods share the same WR core, they can also use the same red/black Seidel step, if desired.

With red/black Seidel, pairs of red/black vertical lines are assigned to each processor. Once the black line solutions of the previous iteration have been communicated to the processors where they are needed, all of the red lines can be solved in parallel, with no other synchronization. Similarly, once the red lines have all been solved, and their solutions have been communicated to the processors where they are needed, all of the black lines can be solved in parallel, with no other synchronization. As shown in Fig. 1, in each iteration of the parallel algorithm, the $N/2$ compute nodes first solve the $N/2$ red lines concurrently, and then solve the $N/2$ black lines concurrently. Note that the red line of compute node i has the black line of compute node $i-1$ as its left boundary condition. Accordingly, the first step of the red line solution process is for each compute node i to receive the black line waveform solution sent by compute node $i-1$. The black line that is the right boundary condition of the red line of compute node i is already resident within the node, so that the compute node can now solve its red line. Once its red line is solved, each compute node immediately sends its red line waveform solution to the left. This completes the first half of the iteration: the solution and communication of the red lines.

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

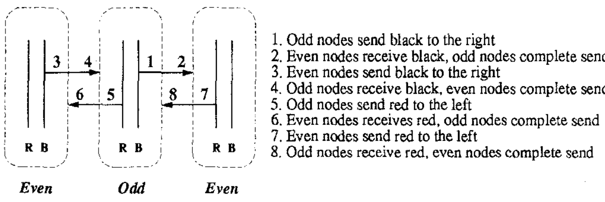
3
4
1
2
6
5
8
7
R B
R B
R B
Even
Odd
Even
1. Odd nodes send black to the right
2. Even nodes receive black, odd nodes complete send
3. Even nodes send black to the right
4. Odd nodes receive black, even nodes complete send
5. Odd nodes send red to the left
6. Even nodes receives red, odd nodes complete send
7. Even nodes send red to the left
8. Odd nodes receive red, even nodes complete send

Fig. 1. Illustration of the communication and computation operations performed by compute node i during one parallel red/black Seidel waveform relaxation step.

If fewer than $N/2$ compute nodes are available, then each compute node is given multiple pairs of red/black lines that are adjacent to each other in the device mesh. Thus, some of the lines (both red and black) residing on a compute node will depend only on other lines residing on that compute node and communication and computation can be overlapped in this case. The red lines that do not depend on solutions from other nodes can be solved before waiting to receive the black line solutions. Similarly, the black lines that do not depend on other compute node solutions are solved before waiting to receive the red line solutions. An outline of parallel red/black block Seidel WR with overlapped communication and computation is shown in Algorithm 4.1.

Algorithm 4.1 (Parallel Red/Black Block WR)

```
1) Choose: Initial guess waveforms that satisfy initial conditions.
2) Iterate:
    a) Post: Nonblocking receive of the black line solution from the left, a nonblocking receive of the red line solution from the right, and nonblocking send of the black line solution to the right.
    b) Solve: The device equations (1) for interior red lines.
    c) Wait: For completion of receipt of black line.
    d) Solve: The device equations (1) for the remaining red line when requisite black line is received.
    e) Post: Nonblocking send of the red line solution to the left.
    f) Solve: The device equations (1) for interior black lines.
    g) Wait: For completion of sends and receipt of red line.
    h) Solve: The device equations (1) for the remaining black line when requisite red line is received.
```

Parallel Pointwise Newton-CGS

In its pointwise approach, the pWORDS program uses a hybrid Newton–Krylov algorithm [28], in which a preconditioned iterative solver is used to solve the linear systems arising at each Newton iteration of each timestep of an implicit integration formula applied to (1). Stabilized variants of the conjugate-gradient squared (CGS) algorithm [25], [31], [32] are the most popular iterative methods for device simulation and they have proved to be the most effective serial algorithm (as well as the most effective parallel pointwise algorithm) for the examples presented in this paper. To partition the problem for a parallel implementation, the pointwise Newton-CGS method in pWORDS uses the same vertical-line block

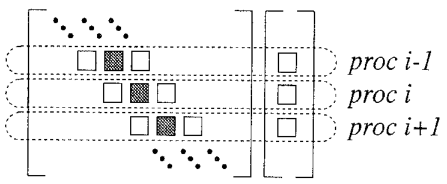

Fig. 2. To partition the matrix-vector product, each processor is assigned the block rows corresponding to a pair of vertical line blocks. The diagonal blocks (used for preconditioning) are shown in grey.

partitioning as in the waveform methods (however, the blocks were processed only in block Jacobi fashion). The blocking is then used to partition the block tridiagonal matrix of the whole problem.

First, the host assigns pairs of vertical-line blocks to the compute nodes exactly as in the waveform method. Given a particular block, the corresponding compute node is responsible for the storage and computation of the corresponding pieces of the matrix and CGS vectors in that "block row," as shown in Fig. 2. For the block tridiagonal matrix of the whole problem, this implies that each compute node must generate and store the appropriate block diagonal pieces of the matrix, as well as the off-diagonal blocks for the block rows. Once a particular block is assigned to it, each compute node is responsible for generating the corresponding piece of the vector resulting from the matrix-vector product. This approach leads naturally to a block Jacobi preconditioner for CGS. That is, the preconditioner is the block diagonal matrix represented by the diagonal blocks on each processor.

Unfortunately, partitioning the matrix and the vectors implies that the parallel pointwise CGS algorithm requires many communication steps, each consisting of relatively small packets. Before every Newton iteration at every timepoint, a compute node must receive the two vectors of solutions of the neighboring lines from the left and the right, in order to generate the block diagonal and block off-diagonal matrices. To accomplish the matrix-vector product for each CGS iteration, a compute node must exchange pieces of the multiplicand vector with the neighboring compute nodes. Moreover, each CGS iteration requires three inner product calculations, each of which requires a global sum. Although it is possible to overlap these communication operations with local computation, a significant amount of interprocessor synchronization is still required. Nonblocking collective operations, such as those proposed for MPI-2 [33], may prove helpful, however.

V. Experimental Results

To compare the parallel performance of the accelerated waveform methods and the pointwise Newton-CGS algorithms, numerical experiments were conducted using eight examples. Instead of plain WR, the more efficient WR Newton (WRN) [6] variant was used. Although it is more efficient, the WRN method is not an *accelerated* waveform method. That is, its convergence rate is the same as that of WR but it takes about 1/3 the work per iteration. Similarly, instead of applying CSOR to plain WR, CSOR was applied to WRN (the resulting method is denoted CSORN to distinguish it from plain CSOR).

TABLE I
Description of Devices Simulated in Numerical Experiments

device	description	mesh	unknowns	interval
ldd	lightly-doped drain	15×20	656	51.2 psec
soi	silicon-on-insulator	18×24	856	51.2 psec
kar	abrupt junction	19×31	1379	512 psec

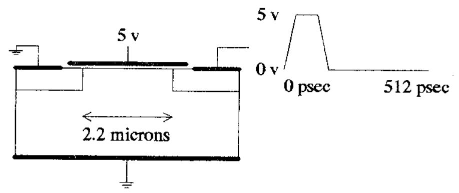

Fig. 3. Illustration of the drain-driven **karD** example.

To provide an initial guess for WRN and CSORN, 16 or 32 initial plain WR iterations were used.

First, the three different MOS device models shown in Table I were used to construct six simulation examples, each device being subjected to either a drain voltage pulse with the gate held high (the **D**-suffix examples), or a gate voltage pulse with the drain held high (the **G**-suffix examples). In addition, to observe the effect on WR convergence, two additional gate-pulsed examples were constructed by refining the device meshes of **karG** and **soiG** to contain 64 vertical lines. All eight examples ranged from low to high drain current, and in the **G** examples, the gate displacement current was substantial because the applied voltage pulses changed at a rate of $.2 \sim 2$ V/ps. Dirichlet boundary conditions were also imposed by ohmic contacts at the source and along the bottom of the substrate, both held at zero volts. Neumann reflecting boundary conditions were imposed along the left and right edges of the meshes. The convergence criterion for all experiments was the requirement that the maximum relative error over the simulation interval in the value of any terminal current be less than 10^{-4}. The drain-driven **karD** test setup is illustrated in Fig. 3.

Although pWORDS supports variable-timestep, variable-order multirate integration (with BDF to order five), to simplify comparisons between the different solution methods, the backward Euler method using 256 fixed timesteps was used for all experiments, on the indicated simulation intervals. Simulations using second-order BDF are given at the end of this section and demonstrate that the results given here are not affected by the order of integration. The use of fixed timesteps avoids the problem of load balancing for the parallel waveform methods, since each vertical-line block will require nearly the same amount of work. At the same time, using global uniform timesteps eliminates the ability of WR to exploit multirate behavior, one of the primary computational advantages of WR on a serial machine. Nevertheless, as the results will show, even without the multirate advantage, the accelerated WR algorithms are competitive on serial machines with the best pointwise methods. Moreover, this is one of the key points of this paper because of their superior scalability, accelerated WR methods are much faster than pointwise methods on parallel processors.

TABLE II
COMPARISON OF SERIAL CPU TIMES REQUIRED FOR POINTWISE METHODS AND WAVEFORM METHODS ON SPARCstation 5

	Pointwise Method		Waveform Methods							
Example	Direct	CGS	WRN	iters	CSOR	iters	WGMRES	iters	WCGS	iters
lddD	298	408	6874	1434	561	100	2476	243	N/A	N/A
lddG	315	384	3195	657	344	55	1383	144	2337	171
soiD	498	172	1717	284	379	50	635	63	532	53
soiG	534	177	1307	209	359	45	778	60	538	53
karD	1718	567	5459	515	1131	92	1747	87	2480	101
karG	1786	753	4694	440	744	55	2322	113	2310	94
soiG64	2614	1724	29697	1885	2165	119	14198	228	4467	129
karG64	4839	3982	40782	1886	3461	141	20264	354	N/A	N/A

TABLE III
COMPARISON OF SERIAL CPU TIMES REQUIRED FOR POINTWISE METHODS AND WAVEFORM METHODS ON RS6000/590

	Pointwise Method		Waveform Methods							
Example	Direct	CGS	WRN	iters	CSOR	iters	WGMRES	iters	WCGS	iters
lddD	65	114	2147	1434	173	100	729	243	N/A	N/A
lddG	68	107	1000	657	107	55	410	144	668	171
soiD	114	49	525	284	115	50	187	63	159	53
soiG	122	51	406	209	111	45	231	60	163	53
karD	405	169	1593	515	326	92	495	87	712	101
karG	422	225	1378	440	215	55	654	113	667	94
soiG64	634	544	9168	1885	660	119	2626	228	1308	129
karG64	1160	1686	11961	1886	995	141	5456	354	N/A	N/A

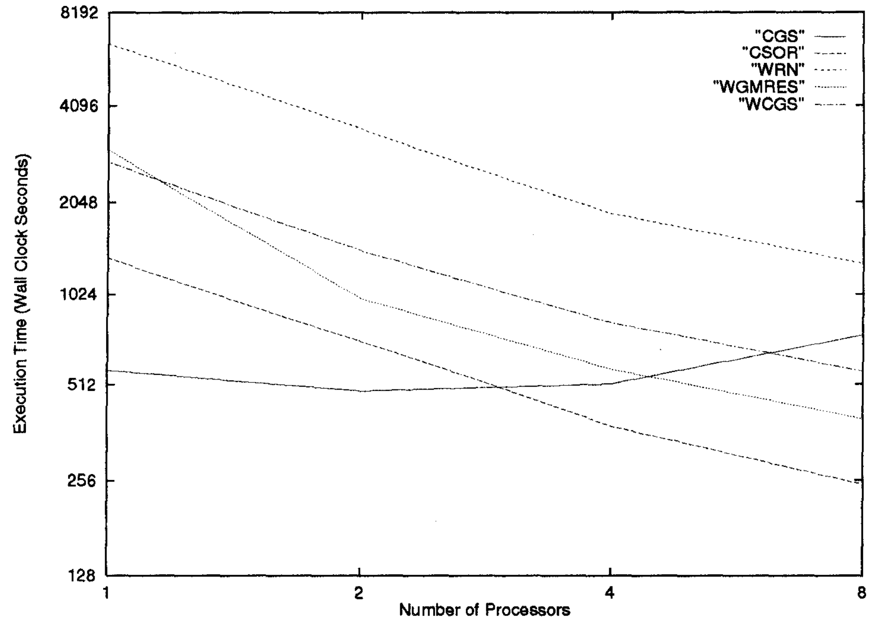

Fig. 4. SPARCstation cluster execution times as a function of number of processors for pointwise and waveform methods for **karD** example.

As a baseline for the parallel comparisons, Tables II and III show the serial CPU times required for solution of the eight examples, for pointwise methods and waveform methods, using a Sun SPARCstation 5 and an IBM RS/6000 Model 590 workstation, respectively. For the pointwise methods, the *Direct* column shows the result of using direct factorization (sparse Gaussian elimination) to solve the matrix problem at each time point and the *CGS* column shows the result of using the iterative CGS algorithm. The waveform method columns show the result of using ordinary Gauss–Seidel WRN, the same algorithm accelerated with CSOR, WGMRES, and WCGS. Note that the CSOR results are competitive with the better pointwise results. For these results, the convolution function $\omega(t)$ was precomputed so the CSOR times do not include the time necessary to compute $\omega(t)$. The N/A in the table for the WCGS results specifies that the method did not converge for the indicated example.

Fig. 4 shows wall-clock execution time as a function of number of processors for pointwise and waveform methods for the **karD** example run on a cluster of SPARCstation 5 workstations. Note that because of virtual memory "thrashing" the single processor wall-clock times can be much higher

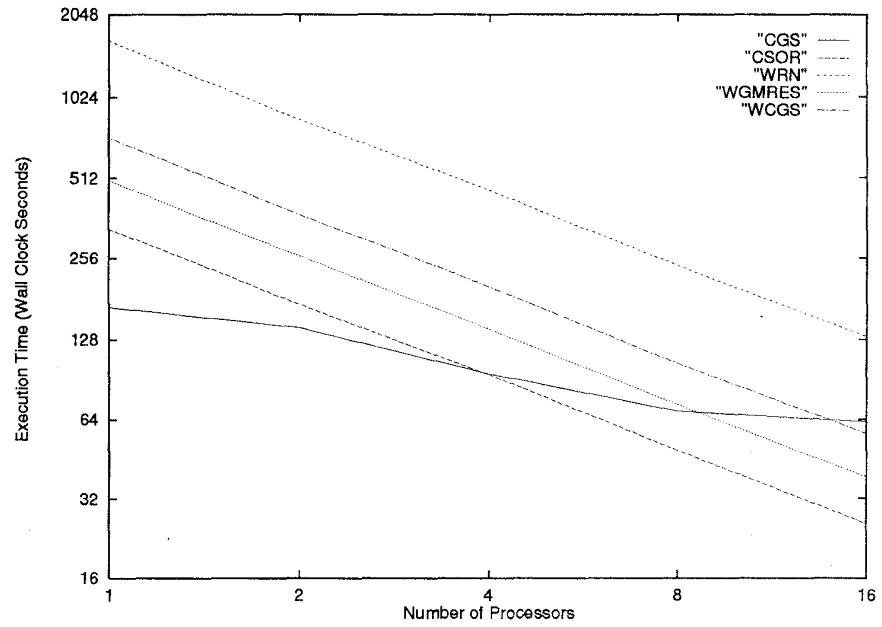

Fig. 5. IBM SP-2 execution times as a function of number of processors for pointwise and waveform methods for **karD** example.

than the single processor CPU times. Due to the high cost of cluster-based communication, parallel speedup for the pointwise method (CGS) is extremely limited (showing an increase on larger numbers of processors). On the other hand, the waveform methods all show a remarkable scalability, achieving a speedup of nearly a factor of six on eight processors.

Fig. 5 shows (wall-clock) execution time as a function of number of processors for pointwise and waveform methods for the **karD** example run on the IBM SP-2. In this case, the pointwise method is able to obtain a speedup in parallel. However, the speedup is limited as the number of processors increases. On the other hand, the waveform methods continue to show a nearly linear speedup up to the number of processors that provide maximum available parallelism (16 in this case for two lines per compute node). The qualitative results on the cluster and the SP-2 are the same. The best pointwise method performs well in serial and on small numbers of processors, but speedup becomes limited due to communication costs. On the other hand, the waveform methods easily surpass the pointwise methods for larger numbers of processors. Note that the problems studied here are of moderate size. Thus, when large numbers of processors are used, the pointwise methods do not have as favorable a computation to communication ratio as do the waveform methods. Hence, the waveform methods exhibit much better scalability.

Tables IV and V summarize the best timing results obtained for each method on the SPARC cluster and on the IBM SP-2, respectively. Note that the waveform methods, in particular, CSOR, are fastest in every case. The superior scalability of the waveform methods is due primarily to the communication and computation structure inherent to the waveform approach. (We remark again that all of the waveform approaches share the same waveform computational core.) Since communication is infrequent and takes place in large packets, the waveform methods are much better able to tolerate latency. Of the

TABLE IV
SUMMARY OF THE BEST TIMING RESULTS FOR EACH METHOD ON THE SPARC CLUSTER

	Pointwise		Waveform				
Example	CGS	procs	WRN	CSOR	WGMRES	WCGS	procs
lddD	414	1	1838	146	560	N/A	10
lddG	387	1	842	73	348	575	10
soiD	173	1	400	75	150	129	12
soiG	179	1	317	72	171	131	12
karD	488	2	1143	231	390	550	8
karG	663	2	966	152	513	510	8
soiG64	1301	2	5384	383	1577	796	8
karG64	3969	4	7090	583	2908	N/A	8

TABLE V
SUMMARY OF THE BEST TIMING RESULTS FOR EACH METHOD ON THE IBM SP-2

Example	CGS	procs	WRN	CSOR	WGMRES	WCGS	procs
lddD	70	10	261	21	89	N/A	10
lddG	77	10	122	12	50	85	10
soiD	30	12	56	12	20	18	12
soiG	30	12	43	12	24	17	12
karD	64	16	128	26	40	57	16
karG	87	16	109	17	52	53	16
soiG64	121	16	380	27	117	52	32
karG64	345	16	471	39	220	N/A	32

waveform methods, the CSOR approach typically offers the best performance. This is due to two factors. First, the CSOR method requires much less work per iteration than do the Krylov-subspace approaches. Second, since the method is designed to give essentially optimal convergence at each temporal frequency, CSOR tends to have faster convergenge than the Krylov-subspace methods. On the other hand, CSOR may not be as widely applicable as the Krylov-subspace methods, but the applicability of the CSOR approach is still an open question.

Finally, to demonstrate that the acceleration properties of the accelerated waveform methods are independent of the order of integration, Table VI shows serial CPU times required for

TABLE VI
COMPARISON OF SERIAL CPU TIMES REQUIRED FOR POINTWISE METHODS AND WAVEFORM METHODS ON RS6000/590 USING SECOND-ORDER BDF

	Pointwise Method		Waveform Methods							
Example	Direct	CGS	WRN	iters	CSOR	iters	WGMRES	iters	WCGS	iters
lddG	568	121	1139	662	233	66	429	144	642	139
soiD	1037	55	590	284	272	52	346	72	381	68
soiG	1039	56	522	208	281	45	330	52	358	54
karD	3612	198	1689	517	463	92	856	97	1364	113
karG	3613	232	1689	441	471	55	915	129	N/A	N/A
karG64	4918	1693	13767	1873	1858	143	9873	479	N/A	N/A

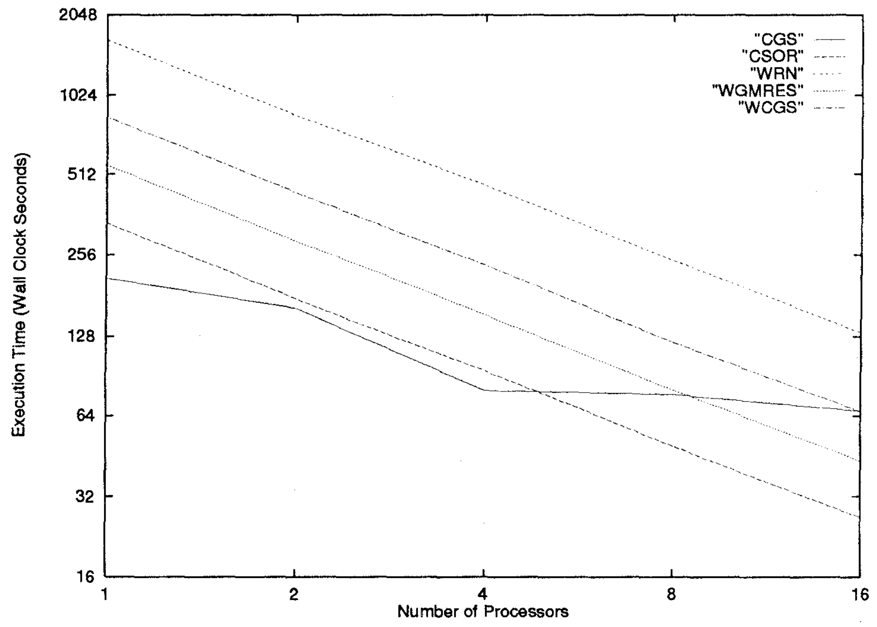

Fig. 6. IBM SP-2 execution times as a function of number of processors for pointwise and waveform methods for **karD** example, using second-order BDF integration.

solution of the eight examples using second-order BDF on an IBM RS/6000 Model 590 workstation. As with first-order BDF, the CSOR results are reasonably competitive with the better pointwise results.

Fig. 6 shows (wall-clock) execution time as a function of number of processors for pointwise and waveform methods for the **karD** example run on the IBM SP-2 using second-order BDF. The results are similar to those obtained with first-order BDF, shown in Fig. 5.

VI. CONCLUSION

For the examples studied in this paper, the comparison of accelerated WR algorithms to pointwise methods showed that accelerated waveform methods are competitive with standard pointwise methods on serial machines and that for line-block domain partitionings, accelerated waveform methods are significantly faster on commonly available loosely coupled MIMD machines. In the hostile communication environment of a university workstation cluster, waveform methods were able to achieve significant parallel speedup. On a dedicated parallel machine, the IBM SP-2, the waveform methods were able to achieve nearly linear speedup. On the other hand, parallel versions of standard pointwise methods exhibited only limited parallel speedup in either parallel environment, though a more careful domain partitioning could improve those results. WR can be viewed as a technique for organizing communication and computation in a parallel environment. Waveform methods perform more computation per number of communication operations than do the pointwise methods and are thus better able to tolerate communication latency. In this sense, WR methods can be viewed as an orthogonal strategy to careful domain partitioning in that they provide another degree of freedom with which to improve parallel efficiency.

The waveform methodologies presented here are general strategies for increasing the parallel performance of time-dependent problems. The results in this paper are a preliminary indication that, as MIMD machines and cluster-based computing become more prevalent, accelerated waveform methods should be seriously considered for those areas of simulation requiring the solution of IVP's. Defining the classes of problems for which waveform methods are most appropriate is an open question (and one which we hope to answer with subsequent studies in this area). Of particular interest is a study of the performance of accelerated waveform methods for device simulations having more sophisticated physics and/or unstructured meshes.

Acknowledgment

The authors wish to thank Professors L. N. Trefethen, D. Rose, P. Lanzcron, A. Toomre, and O. Nevanlinna for many valuable suggestions.

References

[1] C. Gardner, P. J. Lanzkron, and D. J. Rose, "A parallel block iterative method for the hydrodynamic device model," *IEEE Trans. Computer-Aided Design*, vol. 10, pp. 1187–1192, Sept. 1991.

[2] D. Webber, E. Tomacruz, R. Guerrieri, T. Toyabe, and A. Sangiovanni-Vincentelli, "A massively parallel algorithm for three-dimensional device simulation," *IEEE Trans. Computer-Aided Design*, vol. 10, pp. 1201–1209, Sept. 1991.

[3] A. Agarwal, "Limits on interconnection network performance," *IEEE Trans. Parallel Distrib. Sys.*, vol. 2, pp. 398–412, Oct. 1991.

[4] B. P. Herndon, N. R. Aluru, A. Raefsky, R. J. G. Goossens, K. H. Law, and R. W. Dutton, "A methodology for parallelizing PDE solvers: Application to semiconductor device simulation," in *Proc. VIIth SIAM Conf. Parallel Processing Sci. Comput.*, San Francisco, CA, SIAM, Feb 1995, pp. 239–240.

[5] 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.

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

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

[8] R. D. Skeel, "Waveform iteration and the shifted Picard splitting," *SIAM J. Sci. Statist. Comput.*, vol. 10, no. 4, pp. 756–776, 1989.

[9] C. Lubich and A. Osterman, "Multigrid dynamic iteration for parabolic problems," *BIT*, vol. 27, pp. 216–234, 1987.

[10] S. Vandewalle, *Parallel Multigrid Waveform Relaxation for Parabolic Problems*, Teubner-Skripten zur Numerik. Stuttgart, Germany: B. G. Teubner, 1993.

[11] A. Lumsdaine, M. Reichelt, and J. White, "Conjugate direction waveform methods for transient two-dimensional simulation of MOS devices," in *Proc. Int. Conf. Computer-Aided Design*, Santa Clara, CA, Nov. 1991, pp. 116–119.

[12] A. Lumsdaine, "Theoretical and practical aspects of parallel numerical algorithms for initial value problems, with applications," Ph.D. thesis, Mass. Inst. Tech., Cambridge, MA, 1992.

[13] M. W. Reichelt, "Optimal frequency-dependent SOR acceleration of waveform relaxation with application to semiconductor device simulation," in *Proc. Copper Mountain Conf. Multigrid Methods*, Copper Mountain, CO, 1993.

[14] M. Reichelt, J. White, and J. Allen, "Waveform relaxation for transient two-dimensional simulation of MOS devices," in *Proc. Int. Conf. Computer-Aided Design*, Santa Clara, CA, Nov. 1989, pp. 412–415.

[15] R. Bank, W. Coughran, Jr., W. Fichtner, E. Grosse, D. Rose, and R. Smith, "Transient simulation of silicon devices and circuits," *IEEE Trans. Computer-Aided Design*, vol. 4, pp. 436–451, Oct. 1985.

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

[17] R. S. Muller and T. I. Kamins, *Device Electronics for Integrated Circuits*. New York: Wiley, 1986.

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

[19] K. Mayaram and D. Pederson, "CODECS: A mixed-level device and circuit simulator," in *Proc. Int. Conf. Computer-Aided Design*, Santa Clara, CA, Nov. 1988, pp. 112–115.

[20] A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*. Englewood Cliff, NJ: Prentice-Hall, 1989.

[21] D. M. Young, *Iterative Solution of Large Linear Systems*. Orlando, FL: Academic, 1971.

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

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

[24] 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.

[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] A. Lumsdaine and J. K. White, "Accelerating dynamic iteration methods with application to semiconductor device simulation," in *Proc. Copper Mountain Conf. Iterative Methods*, Copper Mountain, CO, Apr. 1992.

[27] R. Saleh and J. White, "Accelerating relaxation algorithms for circuit simulation using waveform-Newton and step-size refinement," *IEEE Trans. Computer-Aided Design*, vol. 9, pp. 951–958, 1990.

[28] 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.

[29] N. E. Doss, W. Gropp, E. Lusk, and A. Skjellum, "An initial implementation of MPI," Math. Comput. Sci. Div., Argonne National Laboratory, Argonne, IL, Tech. Rep. MCS-P393-1193, 1993.

[30] MPI. Forum, "MPI: A Message Passing Interface," in *Proc. Supercomputing*, Nov. 1993, pp. 878–883.

[31] R. W. Freund and N. M. Nachtigal, "A quasiminimal residual method for non-Hermition linear systems," *Numer. Math.*, vol. 60, pp. 315–339, 1991.

[32] R. Freund, "A transpose-free quasiminimum residual algorithm for non-Hermitian linear systems," *SIAM J. Sci. Comput.*, vol. 14, pp. 470–482, 1993.

[33] Message Passing Interface Forum, "MPI-2," July 1995. http://www.mcs.anl.gov/Projects/mpi/mpi2/mpi2.html.

Andrew Lumsdaine (S'90–M'91) received the S.B.E.E., S.M.E.E., and Ph.D. degrees from the Massachusetts Institute of Technology, Cambridge, in 1984, 1986, and 1992, respectively.

He was an engineer in the Manufacturing Development Group with the Packard Electric Division of General Motors durign 1986. He is currently with the faculty of the Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN. His research interests include parallel processing, scientific computing, numerical methods and software, and VLSI circuit and semiconductor device simulation.

Dr. Lumsdaine is a member of SIAM and is an active participant in the MPI Forum. In 1995, he received the NSF Career Development Award.

Mark W. Reichelt (S'88–M'91) received the S.B., S.M., and Ph.D. degrees in electrical engineering and computer science from the Massachusetts Institute of Technology, Cambridge, in 1987 and 1993.

From 1984 to 1990, he held summer positions with AT&T Bell Laboratories and Intel. Since 1993, he has been with The MathWorks, Natick, MA. His research interests include scientific computing, numerical methods, parallel processing, VLSI, and semiconductor device simulation.

Jeffrey M. Squyres received the B.S. degree in computer engineering, and the M.S. degree in computer science and engineering from the University of Notre Dame, Notre Dame, IN, in 1994 and 1996, respectively. He is currently pursuing the Ph.D. degree at Notre Dame in computer science and engineering.

His research interests include high-performance parallel and distributed computing, networking, and scalable software. He is an active participant in the MPI Forum.

Jacob K. White (S'80–M'83) received the B.S. degree in electrical engineering and computer science from the Massachusetts Institute of Technology (M.I.T.), Cambridge, and the S.M. and Ph. D. degree in electrical engineering and computer science from the University of California, Berkeley.

He was with the IBM Thomas J. Watson Research Center from 1985 to 1987, and was the Analog Devices Career Developement Assistant Professor with the M.I.T. from 1987 to 1989. He is currently an Associate Professor with M.I.T. He supervised the development of FASTCAP and FASTHENRY, two widely used signal integrity analysis tools. His current research interests are in serial and parallel numerical algorithms for problems in circuit, interconnect, device, and microelectromechanical system design.

Dr. White was a 1988 Presidential Young Investigator and an Associate Editor for the IEEE Transactions on Computer-Aided Design.