# Massively Parallel Simulation Algorithms for Grid-Based Analog Signal Processors

Andrew Lumsdaine *Member, IEEE,* L. Miguel Silveira, *Student Member, IEEE,* and Jacob K. White, *Member, IEEE*

***Abstract*—This paper presents the algorithms for CMVSIM, a program for performing the transient simulation of grid based analog signal processors on a massively parallel computer. A grid-based equation formulation approach and a block-diagonal preconditioned CGS algorithm are described, and it is shown how they are used to efficiently perform transient simulation using the massively parallel Connection Machine. Experimental results using CMVSIM to simulate realistic image processing circuits are given to demonstrate that the algorithms presented are effective for a general class of grid-based signal processors. In particular, the results presented demonstrate that CMVSIM running on a full-size Connection Machine can be as much as 650 times faster than what is, to the authors' knowledge, the fastest serial transient simulation algorithm running on a SUN-4/490 workstation.**

## I. 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 $256 \times 256$, 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[1] [3].

In this paper, algorithms are presented for simulating grid-based analog signal processors on a massively parallel computer. In Section II, 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 III and the massively parallel implementation of the algorithms is presented in Section IV. Experimental results using the Connection Machine are presented in Section V. Finally, conclusions and suggestions for further work are contained in Section VI.

## II. Problem Description

In this section, we introduce grid-based analog signal processors with a simple example, and then describe a general approach to formulating the equations for grid-based circuits. We will show in Section IV that the formulation approach described below leads to an efficient mapping of the problem onto a massively parallel processor. To end this section, we will show how the formulation approach also makes it possible to easily specify grid-based signal processors to the Connection Machine Vision/SIMulation Program (CMVSIM), by briefly describing the program's input files.

### 2.1. Motivational Problem

Consider the circuit in Fig. 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

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

Manuscript received April 12, 1993. This work was supported by the Defense Advanced Research Projects Agency under Contract N00014-87-K-825 and by the Portuguese INVOTAN committee. This paper was recommended by Associate Editor D. Rose.

A. Lumsdaine was with the Department of Electrical Engineering and Computer Science at the Massachusetts Institute of Technology. He is now with the Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN 46556.

L. M. Silveira and J. K. White are with the Research Laboratory of Electronics, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139.

IEEE Log Number 9212338.

[1]Connection Machine is a registered trademark of Thinking Machines Corporation.

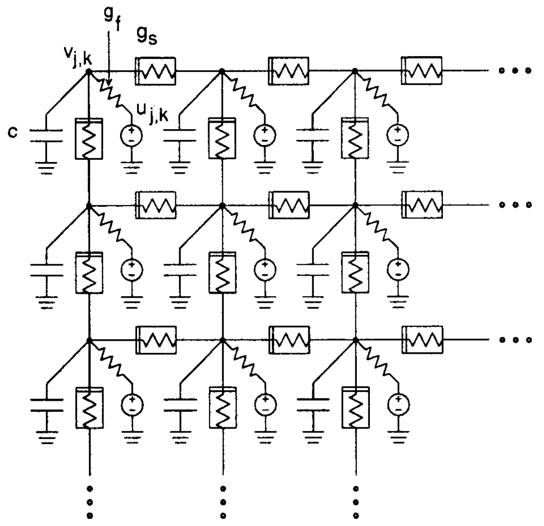


Fig. 1. Grid of nonlinear resistors.

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 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 Fig. 1 are replaced by subcircuits of physical circuit elements. For example, in Mead's Silicon Retina [1], the voltage source $u_{j,k}$ and 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 V).

## 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 Fig. 2. Let the subcircuit have $M_{\text{int}}$ internal nodes and let it connect to internal nodes of its nearest neighbor to the north, east, west, and south with $M_{\text{N}}$, $M_{\text{E}}$, $M_{\text{W}}$, and $M_{\text{S}}$ terminal nodes, 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 provide a means of more formally describing the grid circuit behavior, given the description of the behavior of the individual subcircuits.

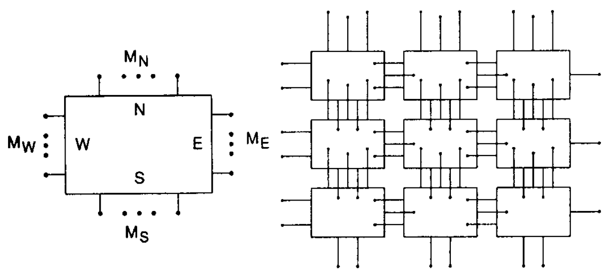


Fig. 2. A single subcircuit, shown here as a multiterminal element, and a grid constructed of such elements by attaching nodes of each element to nodes of its neighbors.

*Proposition 2.1:* Let $\tilde{\mathcal{C}}$ be an $n + m$ node circuit with node voltage vector $\tilde{\mathbf{v}}(t) \in \mathbf{R}^{n+m}$, and node charge and current vectors $\tilde{\mathbf{q}}(\tilde{\mathbf{v}}(t), t), \tilde{\mathbf{i}}(\tilde{\mathbf{v}}(t), t) \in \mathbf{R}^{n+m}$, respectively. Consider a second circuit, $\mathcal{C}$, which is formed from $\tilde{\mathcal{C}}$ by joining each node $j = 1, \cdots, n$ to a set of nodes $K_j \subset \{n + 1, \cdots, n + m\}$, such that $\mathcal{C}$ is a well-defined circuit and has nodal voltage vector $\mathbf{v}(t) \in \mathbf{R}^n$, and nodal charge and current vectors $\mathbf{q}(\mathbf{v}(t), t), \mathbf{i}(\mathbf{v}(t), t) \in \mathbf{R}^n$, respectively. Then, there exists a topological matrix $\mathbf{H}$, such that

$$\mathbf{q}(\mathbf{v}(t), t) = \mathbf{H}^T\tilde{\mathbf{q}}(\mathbf{H}\mathbf{v}(t), t)$$
$$\mathbf{i}(\mathbf{v}(t), t) = \mathbf{H}^T\tilde{\mathbf{i}}(\mathbf{H}\mathbf{v}(t), t).$$

*Proof:* Define $\mathbf{H}: \mathbf{R}^n \to \mathbf{R}^{n+m}$ by:

$$H_{k,j} = \begin{cases} 1 & \text{if } k = j \\ 1 & \text{if node } k \in K_j. \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Obviously, substituting $\mathbf{H}\mathbf{v}(t)$ for $\tilde{\mathbf{v}}(t)$ will insure that all devices in $\tilde{\mathcal{C}}$ have the same terminal voltages as the associated devices in $\mathcal{C}$. Thus, devices contribute the same charge to their terminal nodes whether connected as in $\mathcal{C}$ or as in $\tilde{\mathcal{C}}$. Now consider a component $q_j$ of the vector $\mathbf{q}(\mathbf{v}(t), t)$, corresponding to the sum of charges at node $j$ in $\mathcal{C}$. Node $j$ in $\mathcal{C}$ either corresponds to a single node $j$ in $\tilde{\mathcal{C}}$ or to the several nodes joined with node $j$ when $\mathcal{C}$ is constructed from $\tilde{\mathcal{C}}$. In the former case, $q_j = \tilde{q}_j$, in the latter, $q_j = (\Sigma_{k \in K_j} \tilde{q}_k) + \tilde{q}_j$, and therefore, $\mathbf{q}(\mathbf{v}(t), t) = \mathbf{H}^T\tilde{\mathbf{q}}(\mathbf{H}\mathbf{v}(t), t)$. By an analogous argument, it follows that $\mathbf{i}(\mathbf{v}(t), t) = \mathbf{H}^T\tilde{\mathbf{i}}(\mathbf{H}\mathbf{v}(t), t)$. □

An alternative proof can be constructed using branch incidence matrices [5].

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

$$\mathbf{H} = \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & & 1 \\ & 1 & & \end{bmatrix} \tag{2}$$

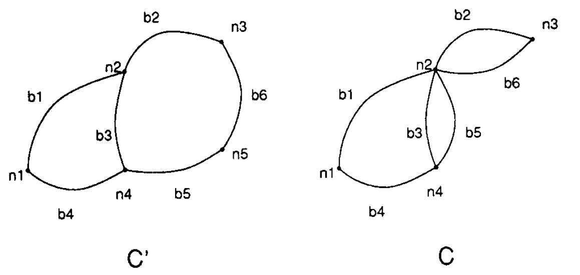


Fig. 3. Example graphs for circuits $\tilde{\mathcal{C}}$ and $\mathcal{C}$. The circuits both have branches b1 through b6; circuit $\tilde{\mathcal{C}}$ has nodes n1 through n5, while circuit $\mathcal{C}$ has nodes n1 through n4. The topological matrix relating the nodes for this example is given in (2).

*Remark:* The matrix $\mathbf{H}$ can be defined more generally by relaxing the condition on the ordering of nodes in Proposition 2.1. That is, $\mathcal{C}$ can be constructed from $\tilde{\mathcal{C}}$ by joining a set of $m$ distinct nodes $\{k_1, \cdots, k_m\} \subset \{1, \cdots, n + m\}$ to the $n$ nodes in $\{1, \cdots, n + m\} \backslash \{k_1, \cdots, k_m\}$, and then renumbering the nodes in $\{1, \cdots, n + m\} \backslash \{k_1, \cdots, k_m\}$ from 1 to $n$. This node renumbering implies that the generalized $\mathbf{H}$ matrix typically will *not* have the diagonal submatrix structure of the $\mathbf{H}$ matrix defined by (1).

In general, an *a priori* decomposition of a grid circuit into identical subcircuits will not be given. In that case, Proposition 2.1 provides a means for describing the construction of a grid circuit from subcircuits or, equivalently, the decomposition of a grid circuit into subcircuits. To apply Proposition 2.1, we note that when separating a grid into identical subcircuits, some nodes of the original grid persist, while others become separated into multiple nodes. Nodes that are not separated in this decomposition are called *internal* nodes. When a node in the grid is separated into a set of nodes in the decomposition, one node from the set will be an internal node, while the others are called *terminal* nodes. In general, it is possible to designate any node in the set to be the internal node. We formalize this approach to equation formulation in the following corollary.

*Corollary 2.2:* Consider an $N \times N$ circuit grid of identical subcircuits, as shown in Fig. 2. Assume each subcircuit has $M_{\text{int}}$ internal nodes plus $M_{\text{term}}$ terminal nodes, ($M_{\text{term}} = M_{\text{E}} + M_{\text{W}} + M_{\text{N}} + M_{\text{S}}$ for the example in Fig. 2), and define $\tilde{M} = M_{\text{int}} + M_{\text{term}}$. Let $\tilde{\mathbf{v}}_\alpha(t) \in \mathbf{R}^{\tilde{M}}$ be the nodal voltage vector for the subcircuit at grid location $\alpha$ when that subcircuit is separated from the grid, and let $\tilde{\mathbf{q}}_\alpha(\tilde{\mathbf{v}}_\alpha(t), t) \in \mathbf{R}^{\tilde{M}}$ and $\tilde{\mathbf{i}}_\alpha(\tilde{\mathbf{v}}_\alpha(t), t) \in \mathbf{R}^{\tilde{M}}$ be the associated nodal charge and current vectors, respectively. Next, define $\tilde{n} = N^2\tilde{M}$, let $\tilde{\mathbf{v}} \in \mathbf{R}^{\tilde{n}}$ represent the nodal voltage vector for the $N^2$ subcircuits separated from each other, and let $\tilde{\mathbf{q}}(\tilde{\mathbf{v}}(t), t) \in \mathbf{R}^{\tilde{n}}$ and $\tilde{\mathbf{i}}(\tilde{\mathbf{v}}(t), t) \in \mathbf{R}^{\tilde{n}}$ be the associated nodal charge and current vectors, respectively. Finally, define $n = N^2M_{\text{int}} + NM_{\text{term}}$, let $\mathbf{v}(t) \in \mathbf{R}^n$ represent the voltage vector for the connected grid circuit, and let $\mathbf{q}(\mathbf{v}(t), t) \in \mathbf{R}^n$ and $\mathbf{i}(\mathbf{v}(t), t) \in \mathbf{R}^n$ be the associated nodal charge and current vectors, respectively.

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

$$\begin{aligned} \mathbf{q}(\mathbf{v}(t), t) &= \mathbf{H}^T\tilde{\mathbf{q}}(\mathbf{H}\mathbf{v}(t), t) \\ \mathbf{i}(\mathbf{v}(t), t) &= \mathbf{H}^T\tilde{\mathbf{i}}(\mathbf{H}\mathbf{v}(t), t). \end{aligned} \tag{3}$$

*Proof:* Order the subcircuit terminals such that nodes $\{1, \cdots, n\}$ correspond only to the internal nodes of the subcircuits and nodes $\{n + 1, \cdots, \tilde{n}\}$ correspond to the terminal nodes of the subcircuits. Apply Proposition 2.1. □

*Remark:* If all $N^2$ subcircuits are identical, then $\tilde{n} = N^2M_{\text{int}} + N^2M_{\text{term}}$ and $n = N^2M_{\text{int}} + NM_{\text{term}}$. However, these formulas for $n$ and $\tilde{n}$ are not usually applicable, because in most circuit examples the subcircuits at the periphery of the grid, denoted as boundary subcircuits, are slightly different from those internal to the grid. This difference in boundary subcircuits has implications for how grid-based circuits are described to the program CMVSIM, and we return to this point in Section 2.3.

*Corollary 2.3:* Let $\mathbf{J}_q = (\partial\mathbf{q}/\partial\mathbf{v})$, $\mathbf{J}_i = (\partial\mathbf{i}/\partial\mathbf{v})$, $\mathbf{J}_{\tilde{q}} = (\partial\tilde{\mathbf{q}}/\partial\mathbf{v})$, and $\mathbf{J}_{\tilde{i}} = (\partial\tilde{\mathbf{i}}/\partial\mathbf{v})$ be the Jacobian matrices for the functions $\mathbf{q}$, $\mathbf{i}$, $\tilde{\mathbf{q}}$, and $\tilde{\mathbf{i}}$ described in Corollary 2.2, respectively, and let $\mathbf{H}$ be the topological matrix relating $\mathbf{q}$ to $\tilde{\mathbf{q}}$ and $\mathbf{i}$ to $\tilde{\mathbf{i}}$. Then, $\mathbf{J}_q$ is related to $\mathbf{J}_{\tilde{q}}$ and $\mathbf{J}_i$ is related to $\mathbf{J}_{\tilde{i}}$ by

$$\begin{aligned} \mathbf{J}_q(\mathbf{v}(t), t) &= \mathbf{H}^T\mathbf{J}_{\tilde{q}}(\mathbf{H}\mathbf{v}(t), t)\mathbf{H} \\ \mathbf{J}_i(\mathbf{v}(t), t) &= \mathbf{H}^T\mathbf{J}_{\tilde{i}}(\mathbf{H}\mathbf{v}(t), t)\mathbf{H} \end{aligned}$$

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

*Proof:* The result follows directly from differentiating (3). □

*Remark:* Computationally, Proposition 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. This should not be surprising to the reader. In a nodal formulation, the nodal charges and currents are calculated by evaluating each device in the circuit solely as a function of its terminal voltages. The above results provide some formal means for extending this type of device evaluation process to a parallel implementation.

Similar results to Corollary 2.2 and Corollary 2.3 can be constructed for other types of grid circuits as well (such as hexagonal grids).

## 2.3. *Describing Arrays to CMVSIM*

A decomposition of the example circuit shown in Fig. 1 is shown in Fig. 4. In this decomposition, 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. Note that this decomposition is not unique. The basic subcircuit for other decompositions can be obtained by rotating the subcircuit in Fig. 4 in the plane with 90 degree incre-

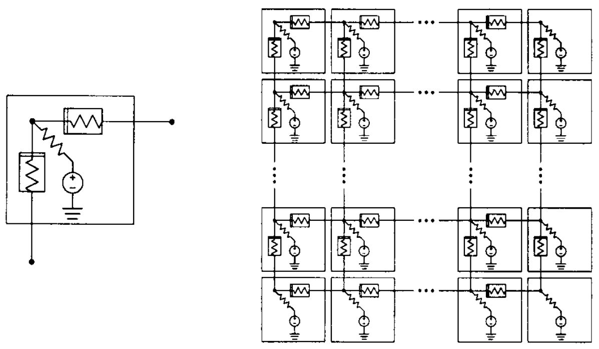


Fig. 4. Decomposition of the example circuit shown in Fig. 1. Note that the subcircuits on the east and south borders of the grid differ from the subcircuits elsewhere in the grid.

ments. Each rotation would thus provide a new decomposition.

This simple example also makes clear that boundary subcircuits differ from the subcircuits internal to the grid. A simple, yet flexible, approach to describing how to modify boundary subcircuits is to subdivide subcircuits into five smaller sub-subcircuits, denoted `here`, `north`, `south`, `east`, and `west`. Subcircuits internal to the grid contain all five sub-subcircuits, and all subcircuits contain the `here` sub-subcircuit. Subcircuits on the north border contain all but the `north` sub-subcircuit, and the analogous sub-subcircuits are deleted from subcircuits on the other three borders. Corner subcircuits, of course, contain only the appropriate three sub-subcircuits.

The sub-subcircuits are described to the program CMVSIM as five separate files. Each of the files contains a complete circuit given in SIMLAB circuit syntax (a slight variant of the SPICE language [6]). The files are denoted by their relationship to the border of the grid, i.e., `north`, `east`, `west`, or `south`; and as mentioned above, the subcircuit circuitry 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). Since the subcircuit description is divided into separate 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** ⟨node1⟩ [⟨node2⟩ · · ·]

Note that when describing a subcircuit to CMVSIM, it is not necessary to explicitly distinguish subcircuit internal nodes from terminals. Instead, special circuit elements, referred to as connector elements, are used to indicate how nodes from one subcircuit are connected to neighboring subcircuit nodes. That the "connection" is from a terminal of one subcircuit to the internal node of another is inferred by the CMVSIM program. Frequently, common nodes correspond to internal nodes to which terminals from neighboring subcircuits are joined.

| lres.rel.e | lres.rel.s | lres.rel.h |
|---|---|---|
| global 0 | global 0 | global 0 |
| r1 here east r r=1k | r1 here south r r=1k | ; common here |
| c0 east here connector | c0 south here connector | r1 here 1 r r=1k |
| | | v1 1 0 dc v=2 |

Fig. 5. Example files for specifying a resistive circuit grid.

As a complete example, the contents of the circuit files to specify a resistive grid are shown in Fig. 5 as the files `lres.rel.e`, `lres.rel.s`, and `lres.rel.h`. Refer to the "CMVSIM users' guide" [7] for more details on using CMVSIM.

## III. Numerical Algorithms

The system of equations that describes an $N \times N$ grid circuit, constructed as in Corollary 2.2, can be written compactly as

$$\frac{d}{dt}\mathbf{q}(\mathbf{v}(t), t) + \mathbf{i}(\mathbf{v}(t), t) = 0. \tag{4}$$

Here, $\mathbf{v}(t)$, $\mathbf{q}(\mathbf{v}(t), t)$, $\mathbf{i}(\mathbf{v}(t), t) \in \mathbf{R}^n$ 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.

The transient simulation of the analog grid involves numerically solving (4). Discretizing (4) with the trapezoidal rule leads to the following algebraic problem for each time step $h$:

$$\frac{2}{h}[\mathbf{q}(\mathbf{v}(t+h), t+h) - \mathbf{q}(\mathbf{v}(t), t)] + [\mathbf{i}(\mathbf{v}(t+h), t+h) + \mathbf{i}(\mathbf{v}(t), t)] = 0. \tag{5}$$

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

$$\mathbf{J}_F(\mathbf{v}^m(t+h), t+h)[\mathbf{v}^{m+1}(t+h) - \mathbf{v}^m(t+h)] = -\mathbf{F}(\mathbf{v}^m(t+h), t+h) \tag{6}$$

where

$$\mathbf{F}(\mathbf{v}(t+h), t+h) = \frac{2}{h}[\mathbf{q}(\mathbf{v}(t+h), t+h) - \mathbf{q}(\mathbf{v}(t), t)] + [\mathbf{i}(\mathbf{v}(t+h), t+h) + \mathbf{i}(\mathbf{v}(t), t)] \tag{7}$$

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

$$\mathbf{J}_F(\mathbf{v}(t+h), t+h) = \frac{2}{h}\frac{\partial\mathbf{q}(\mathbf{v}(t+h), t+h)}{\partial\mathbf{v}} + \frac{\partial\mathbf{i}(\mathbf{v}(t+h), t+h)}{\partial\mathbf{v}}. \tag{8}$$

In "classical" circuit simulators such as SPICE [6], the linear system of equations for each Newton iteration is

TABLE I
COMPARISON OF SERIAL EXECUTION TIME FOR THE TRANSIENT SIMULATION OF THE CIRCUIT IN FIG. 1, WHEN USING DIRECT, AND CONJUGATE-DIRECTION LINEAR SYSTEM SOLVERS[2]

| Size | Direct | CG | ILUCG | CGS | ILUCGS |
|---|---|---|---|---|---|
| 16 × 16 | 11.2 | 18.4 | 23.0 | 34.2 | 27.4 |
| 32 × 32 | 231 | 197 | 148 | 395 | 268 |
| 64 × 64 | 4480 | 2260 | 1550 | 4910 | 2500 |
| 128 × 128 | 82000 | 24200 | 13900 | 51700 | 27400 |
| 16 × 16 | 0.010 | 0.019 | 0.014 | 0.032 | 0.024 |
| 32 × 32 | 0.104 | 0.092 | 0.065 | 0.185 | 0.115 |
| 64 × 64 | 1.04 | 0.530 | 0.366 | 1.22 | 0.609 |
| 128 × 128 | 11.7 | 3.41 | 2.01 | 7.19 | 3.95 |

[2]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. The first chart shows the CPU seconds required for all linear system solutions during the entire simulation; the second chart shows the average number of CPU seconds required for each linear system solution. All execution times are CPU seconds for an IBM RS/6000 model 530 workstation with 128 megabytes of memory.

solved by some form of sparse Gaussian elimination. Sparse Gaussian elimination is an effective strategy for typical large circuits, because such circuits generate Jacobians which are extremely sparse, and have an almost tree-like associated graph. Therefore, these Jacobians can be factored with very little fill-in, and experimental data suggests that for these typical circuits, the number of operations required to compute a factorization increases nearly linearly with the number of circuit nodes [8]. However, problems like the grid circuit in Figure 1 generate Jacobians which, when factored, generate substantially more fill-in than is typical for large circuit problems. In fact, it is well-known that such Jacobians require order $n^{3/2}$ operations to factor, where $n$ is the number of nodes in the grid [9].

Reducing serial computational complexity is only one reason for considering an iterative matrix solution method rather than sparse Gaussian elimination for our class of circuit problems. Since the goal was to develop a simulator which could efficiently exploit a massively parallel SIMD machine, iterative methods are more appealing because they are more easily parallelized on SIMD machines than, for example, parallel nested dissection [10]. In addition, iterative methods like conjugate-direction algorithms [11] are well-suited to solving our class of problems, because only low accuracy is required, and an effective preconditioner is easily extracted from the problem structure (see Section 4.3). Also, if timesteps are made small enough, convergence can be further improved. In that case, the capacitive portion of the Jacobian in (8) will dominate, and it is typically better conditioned and closer to normal than the conductive portion (see [12]).

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), methods suitable for non-symmetric matrices must be considered, e.g., CGNR, GCR, GCR(k), ORTHOMIN, CGS [11], [13]. The present discussion will be restricted to CGS, presented in [13], since experience has shown that it is the most efficient of the methods examined (see also [14], [15]).

To demonstrate the effectiveness of the conjugate-direction algorithm, in Table I the CPU time required to compute the transient analysis of the network in Fig. 1 is compared using several different matrix solution algorithms to solve (6). These results were obtained using the simulation program and experimental setup described in Section V. The conjugate-direction methods shown in Table I are conjugate gradient, conjugate gradient squared (CGS), incomplete LU preconditioned CG (ILUCG), and incomplete LU preconditioned CGS (ILUCGS). The first chart shows the CPU seconds required for all linear system solutions during the entire simulation; the second chart shows the average number of CPU seconds required for each linear system solution. The table does not include the time required to perform the initial ordering and symbolic factorization of the matrix.

The total linear system solution time in Table I is *not* particularly useful for determining the solution time growth with problem size; the number of timesteps, and therefore the number of system solutions, is also increasing with problem size. Instead, consider comparing the different methods using the average CPU time per linear system solution. These average times indicate that solution by sparse Gaussian elimination is much slower than the conjugate-direction algorithms, especially for the larger problem sizes. As can be determined from the table, the computation time growth with problem size using sparse Gaussian elimination is approximately $\mathcal{O}(N^{1.7})$ (for $N$ circuit nodes), and correspondingly, the computation time growth for ILUCG is approximately $\mathcal{O}(N^{1.25})$.

It has been shown both experimentally and theoretically that the cost of sparse Gaussian elimination applied to square grid problems grows as $\mathcal{O}(N^{1.5})$ [9]. The somewhat faster growth of factorization time with problem size demonstrated in the above table may be due to the fact that the Sparse 1.3 package from the Berkeley SPICE3 program [16], [17] is not optimized to best exploit a workstation's cache. Therefore, the percentage of memory references which generate cache misses almost certainly increases with problem size, and this could easily explain the faster than expected growth in factorization time. It should also be noted as *coincidental* that the measured computation time growth of ILUCG for the grid circuit problem matches the theoretical result for a 2-D discretized Laplacian (see [18]). The conductance and capacitance to ground in the circuit implies that the linear system at each timestep is more diagonally dominant than the 2-D discretized Laplacian, and therefore ILUCG converges faster for these examples. That this accelerated convergence doesn't reflect itself in reduced computation time we again attribute to reduced cache performance with increasing problem size.

Finally, note that this example problem has a symmetric Jacobian, and therefore the more efficient CG algorithm can be used instead of CGS. However, the more realistic array examples in the next section use transistors, and therefore do not have symmetric Jacobians. For those examples CGS can not be replaced by CG.

## IV. 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 yielded only limited success [12], [19]. 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.

### 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 II. The problem is mapped onto an $N \times N$ processor array by assigning the data for one subcircuit to each processor (see Fig. 6).

The characteristics of the grid circuit can be obtained with this mapping by using Corollary 2.2, as described in

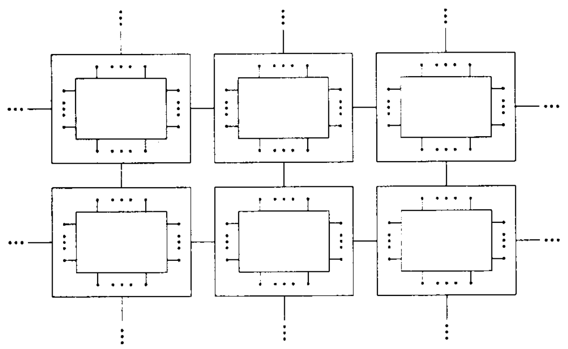


Fig. 6. Mapping of subcircuits to processor grid. Each processor, represented by a single box, contains the data necessary for simulating a single subcircuit. The lines connecting the boxes represent the inter-processor communication network.

Sections 4.2 and 4.3. Note that $\tilde{\mathbf{v}}$, $\tilde{\mathbf{q}}(\tilde{\mathbf{v}})$, and $\tilde{\mathbf{i}}(\tilde{\mathbf{v}})$ can be completely represented by keeping $(M_{\text{int}} + M_{\text{term}})$ components of these vectors on each processor. However, representing $\mathbf{v}$, $\mathbf{q}(\mathbf{v})$, and $\mathbf{i}(\mathbf{v})$ requires only $M_{\text{int}}$ components of each vector on each processor. Therefore, the convention is adopted that the components of $\tilde{\mathbf{v}}$, $\tilde{\mathbf{q}}(\tilde{\mathbf{v}})$, and $\tilde{\mathbf{i}}(\tilde{\mathbf{v}})$ which correspond to the internal nodes are stored in locations $\{1, \cdots, M_{\text{int}}\}$ and the components which correspond to terminal nodes are stored in locations $\{M_{\text{int}} + 1, \cdots, M_{\text{int}} + M_{\text{term}}\}$. With this convention, the vectors $\mathbf{v}$, $\mathbf{q}(\mathbf{v})$, and $\mathbf{i}(\mathbf{v})$ are simply the first $M_{\text{int}}$ locations of the vectors $\tilde{\mathbf{v}}$, $\tilde{\mathbf{q}}(\tilde{\mathbf{v}})$, and $\tilde{\mathbf{i}}(\tilde{\mathbf{v}})$, respectively.

### 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 (7), i.e., computing $\mathbf{F}$ and $\mathbf{J}_F$ as in (7) and (8). Since the linear system solver is a conjugate-direction iterative method, $\mathbf{J}_F$ is not needed explicitly, rather, it is only necessary to have the result of the matrix-vector product $\mathbf{y} = \mathbf{J}_F\mathbf{x}$. If (by definition) $\mathbf{J}_{\tilde{F}} = (2/h)\,\mathbf{J}_{\tilde{q}} + \mathbf{J}_{\tilde{i}}$, then by Corollary 2.3, the result of the matrix–vector product $\mathbf{y} = \mathbf{J}_F\mathbf{x}$ can be calculated according to $\mathbf{y} = \mathbf{H}^T\,\mathbf{J}_{\tilde{F}}\mathbf{H}\mathbf{x}$. Therefore, only $\mathbf{J}_{\tilde{F}}$ is calculated during the device evaluation process.

Using the topological matrix described in Corollary 2.3, the computation and communication required to compute $\mathbf{F}$ and $\mathbf{J}_{\tilde{F}}$ can be described by the following, where the dependence on $t$ is omitted for clarity:

1) Calculate $\tilde{\mathbf{v}} = \mathbf{H}\mathbf{v}$. In this step, the values of $\tilde{\mathbf{v}}$ corresponding to terminal nodes are set with nearest-neighbor communication operations.
2) Calculate the Jacobian, $\mathbf{J}_{\tilde{F}}(\tilde{\mathbf{v}}) = (2/h)\,\mathbf{J}_{\tilde{q}}(\tilde{\mathbf{v}}) + \mathbf{J}_{\tilde{i}}(\tilde{\mathbf{v}})$ and "right-hand side" components, $\tilde{\mathbf{q}}(\tilde{\mathbf{v}})$ and $\tilde{\mathbf{i}}(\tilde{\mathbf{v}})$, by evaluating the cell devices in parallel.
3) Calculate $\mathbf{q}(\mathbf{v}) = \mathbf{H}^T\tilde{\mathbf{q}}(\tilde{\mathbf{v}})$ and $\mathbf{i}(\mathbf{v}) = \mathbf{H}^T\tilde{\mathbf{i}}(\tilde{\mathbf{v}})$. In this step, the values of $\tilde{\mathbf{q}}(\tilde{\mathbf{v}})$ and $\tilde{\mathbf{i}}(\tilde{\mathbf{v}})$ corresponding to terminal nodes are communicated with nearest-neighbor communication operations and added into the appropriate locations of $\mathbf{q}(\mathbf{v})$ and $\mathbf{i}(\mathbf{v})$.

An explicit representation of $\mathbf{H}$ 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.

## *4.3. Linear System Solution*

There are two parts of a conjugate-direction 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 $\mathbf{y} = \mathbf{J}_F\mathbf{x}$ is computed according to $\mathbf{y} = \mathbf{H}^T\mathbf{J}_{\tilde{F}}\mathbf{H}\mathbf{x}$, using the result of Corollary 2.3, with the following sequence of operations:

1) Calculate $\tilde{\mathbf{x}} = \mathbf{H}\mathbf{x}$.
2) Perform parallel block matrix–vector multiplication, $\tilde{\mathbf{y}} = \mathbf{J}_{\tilde{F}}\tilde{\mathbf{x}}$.
3) Calculate $\mathbf{y} = \mathbf{H}^T\tilde{\mathbf{y}}$.

Here, $\mathbf{x}$, $\tilde{\mathbf{x}}$, $\mathbf{y}$, and $\tilde{\mathbf{y}}$ are stored and $\mathbf{Hx}$ and $\mathbf{H}^T\tilde{\mathbf{y}}$ 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 $\mathbf{J}_{\tilde{F}}$ is block diagonal.

*4.3.1. 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 $\mathbf{Ax} = \mathbf{b}$ directly, the conjugate-direction method is applied to the equivalent system

$$\mathbf{P}^{-1}\mathbf{A}\mathbf{x} = \mathbf{P}^{-1}\mathbf{b}.$$

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

For array processors of the form of Fig. 2, the structure of the Jacobian matrix $\mathbf{J}_F$ is that of a block diagonal matrix with block off-diagonal bands. The diagonal blocks are of size $M_{\text{int}} \times M_{\text{int}}$; the block off-diagonal bands are of size $M_{\text{N}} \times M_{\text{N}}$, $M_{\text{E}} \times M_{\text{E}}$, $M_{\text{W}} \times M_{\text{W}}$, and $M_{\text{S}} \times M_{\text{S}}$. This suggests a block iterative method for solving (6), using the diagonal blocks of $\mathbf{J}_F$ 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 $\mathbf{J}_{\tilde{F}}$ is already block diagonal, inverting its blocks is not the same as inverting the block diagonal portion of $\mathbf{J}_F$.

Rather, let $\mathbf{J}_F = \mathbf{P} + \mathbf{R}$, where $\mathbf{P}$ is the block diagonal part of $\mathbf{J}_F$. To use $\mathbf{P}$ as a preconditioner in the CGS algorithm, it must be formed from $\mathbf{J}_{\tilde{F}}$. Let $\tilde{R}_{j,k}$ be the part of $\mathbf{J}_{\tilde{F}}$ corresponding to the coupling between internal and terminal nodes of the same subcircuit—this coupling will become the off-diagonal coupling blocks in $\mathbf{J}_F$. Define $\tilde{\mathbf{P}}$ by $\mathbf{J}_{\tilde{F}} = \tilde{\mathbf{P}} + \tilde{\mathbf{R}}$ (see Fig. 7). Using Corollary 2.3, note that

$$\mathbf{P} = \mathbf{H}^T\tilde{\mathbf{P}}\mathbf{H}. \tag{9}$$

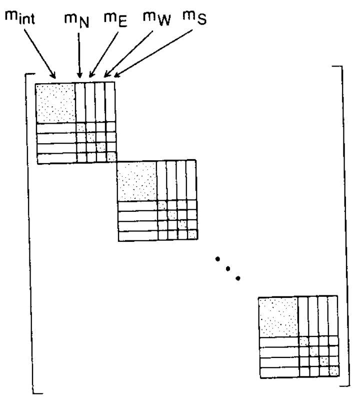


Fig. 7. Definition of $\tilde{\mathbf{P}}$. Since the coupling between internal nodes and terminal nodes will become off-diagonal blocks in $\mathbf{J}_F$, these coupling elements are removed from $\mathbf{J}_{\tilde{F}}$ to form $\tilde{\mathbf{P}}$.

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

1) Form $\mathbf{P} = \mathbf{H}^T\tilde{\mathbf{P}}\mathbf{H}$.
2) Solve $\mathbf{Pz} = \mathbf{r}$.

## V. 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 the SIMLAB program—a SPICE-like circuit simulator developed at MIT [20], [21]—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) [22]. 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 (CM) 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 im-

TABLE II
TOLERANCES USED IN SIMULATIONS REPORTED IN SECTION V

| Tolerance | Relative | Absolute |
|---|---|---|
| Local Truncation Error | $1 \times 10^{-2}$ | $5 \times 10^{-3}$ |
| Newton-Raphson voltage | $1 \times 10^{-3}$ | $1 \times 10^{-3}$ |
| Newton-Raphson current | $1 \times 10^{-3}$ | $1 \times 10^{-9}$ |
| Linear system (current) | $1 \times 10^{-4}$ | $1 \times 10^{-10}$ |

TABLE III
EXPERIMENTAL RESULT: LINEAR CIRCUIT GRID

| | Serial | | | |
|---|---|---|---|---|
| Size | Direct | CG | ILCG | CM 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 | $(1.63 \times 10^4)$ | (531) | (501) | 9.54 |
| 256 × 256 | $(2.01 \times 10^5)$ | (2236) | (2220) | (10) |

age I/O support added. The parallel experiments were run using the CMVSIM program [23], [7]. Implementation of the CMVSIM program required major enhancements 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 [24], [25]. 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.

Both serial and parallel transient simulations used trapezoidal integration with local truncation error (LTE) timestep control. The LTE tolerances and the convergence tolerances for the Newton-Raphson and linear solution iterations are shown in Table II. The serial direct method solver used sparse Gaussian elimination with Markowitz ordering and diagonal pivoting. The CPU time required for the initial ordering and symbolic factorization are not included in the reported linear solution times.

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.

Because some of the larger grid problems either required too much memory, or simply took too long, some of the serial times were extrapolated. To perform the extrapolation, it was assumed that each increase by a factor of two in array dimension resulted in a fixed ratio increase in total simulation time. This ratio was then determined using simulation times for the smaller, more quickly simulated, arrays. Note also, it was not possible to get access to a full-size CM-2, and such a machine would have been necessary to achieve peak efficiency in simulating the largest grid size (256 × 256). For this reason, the simulation times for these largest grids where estimated as well.

### *5.1. Linear Resistive Grid*

Table III shows the results obtained while simulating the linear resistive circuit grid shown in Fig. 8. The circuit for which the results are shown had a 1 kΩ resistance between neighboring nodes, a 33.33 kΩ resistance to ground, and a parasitic capacitance of $1 \times 10^{-15}$ F at each node. A random image was introduced to the network and the startup transient of the first $1 \times 10^{-5}$ s 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.

### *5.2. Nonlinear Resistive Grid*

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

$$i(v) = \frac{\alpha v}{1 + e^{-\beta(\gamma - \alpha v^2)}}. \tag{10}$$

The circuit for which the results are shown had a resistance of 10 kΩ to ground, a parasitic capacitance of $1 \times 10^{-7}$ F and parameter values for the nonlinear resistance of $\alpha = 1 \times 10^{-3}$, $\gamma = 2 \times 10^{-5}$ with $\beta$ being swept from 0 to $1 \times 10^5$ 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 $\beta$ was increased from 0 to $1 \times 10^5$. The reasons for this type of simulation are explained in detail in [26].

For the 128 × 128 grid, the CM achieved a speedup of about a factor of 50. If the serial and parallel results are

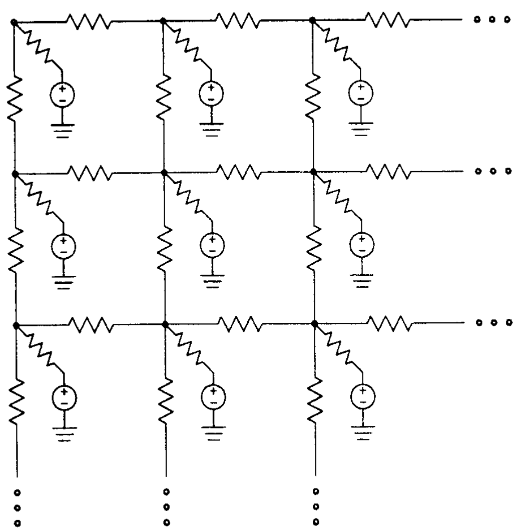


Fig. 8. Linear resistive grid.

TABLE IV
EXPERIMENTAL RESULT: NONLINEAR CIRCUIT 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 | ($8.86 \times 10^4$) | 14 287.90 | 10 377.90 | 896.01 |
| 128 × 128 | ($1.95 \times 10^6$) | ($1.13 \times 10^5$) | ($7.45 \times 10^4$) | 1445.12 |
| 256 × 256 | ($4.28 \times 10^7$) | ($8.99 \times 10^5$) | ($5.35 \times 10^5$) | (2330) |

TABLE V
EXPERIMENTAL RESULT: MEAD'S SILICON RETINA WITH CONSTANT INPUT IMAGE

| | 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 | (21 790) | (35 525) | (10 596) | 1020.88 | 461.42 |
| 128 × 128 | (141 583) | (153 142) | (44 336) | 1048.25 | 463.89 |
| 256 × 256 | (919 959) | (660 172) | (185 510) | (1076) | (466) |

extrapolated to a 256 × 256 grid, the CM should be able to achieve a speedup of over a factor of 200.

### 5.3. *Mead's Silicon Retina*

Tables V and VI show the results obtained while simulating Mead's Silicon Retina as described in [1] and shown in Fig. 9. Table V shows results obtained when presenting the circuit with a constant image while Table VI 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 plain CGS for all grid sizes. For the examples with the random input image, the block diagonal preconditioner reduces 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

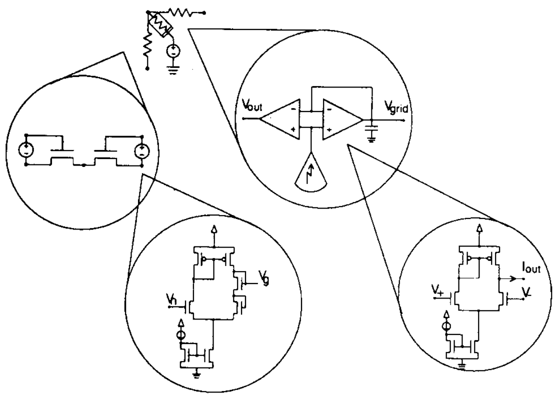


Fig. 9. Mead's Silicon Retina. The simulated chip contains 23 MOS level 3 devices per cell for a total of 376 320 devices and 261 888 nodes in a 128 × 128 grid.

TABLE VI
EXPERIMENTAL RESULT: MEAD'S SILICON RETINA WITH RANDOM INPUT IMAGE

| | Serial | | | CM | |
|---|---|---|---|---|---|
| Size | Direct | CGS | ILCGS | CGS | PCGS |
| 16 × 16 | 1239.50 | 4405.77 | 1244.75 | 1845.34 | 894.42 |
| 32 × 32 | 10 713.1 | 25 740.9 | 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) |

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.

### *5.4. 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 [26] for detailed treatment of the continuations). The network used is shown in Fig. 1 and uses the nonlinear element described by (10); let the linear conductance to ground be called $\lambda_f$. Figure 10 shows a 256 × 256 image—a portion of the San Francisco sky line—used as the input image.

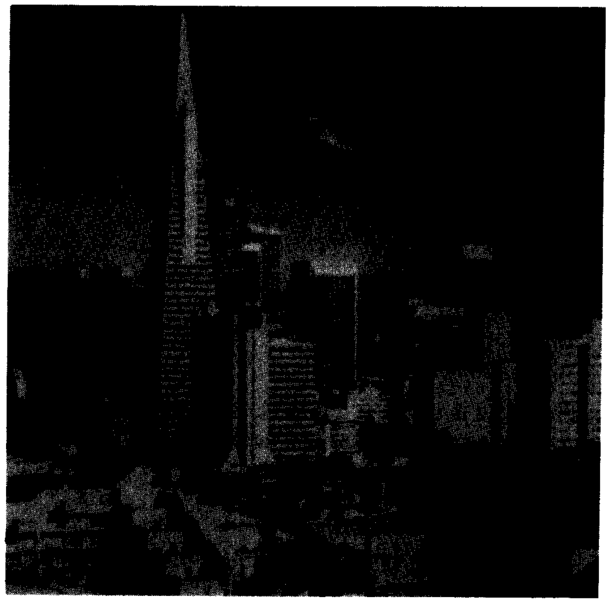
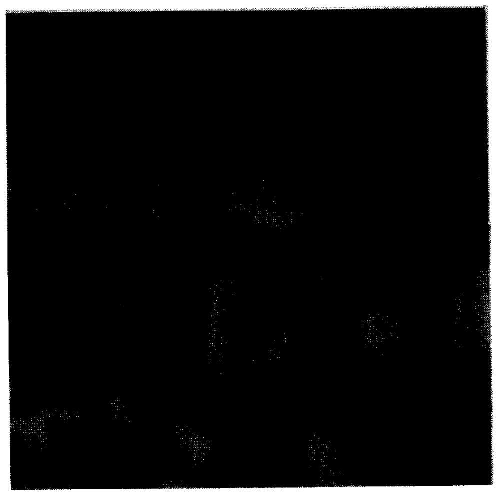


Fig. 10. 256 × 256 image of the San Francisco sky line.

Fig. 11 shows the output produced by the idealized nonlinear network with a continuation performed on the value of $\beta$. The fixed parameter values were $\alpha = 1 \times 10^{-3}$, $\gamma = 1 \times 10^{-5}$, $\lambda_f = 3 \times 10^{-5}$; the value of $\beta$ was linearly varied as a function of time from $\beta = 0$ to $1 \times 10^6$. Fig. 11(a) shows the output of the network at the beginning of the continuation when $\beta = 0$; Fig. 11(b) shows the output of the network at an intermediate point of the continuation when $\beta = 2 \times 10^4$; Fig. 11(c) shows the output of the network at the end of the continuation when $\beta = 1 \times 10^6$.

Fig. 12 shows the output produced by the idealized nonlinear network with a continuation performed on the


(a)

(b)

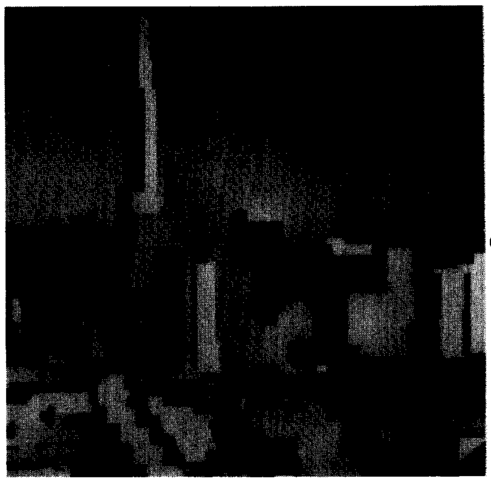
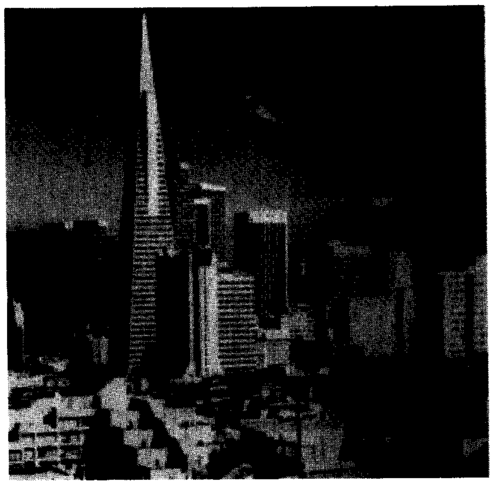
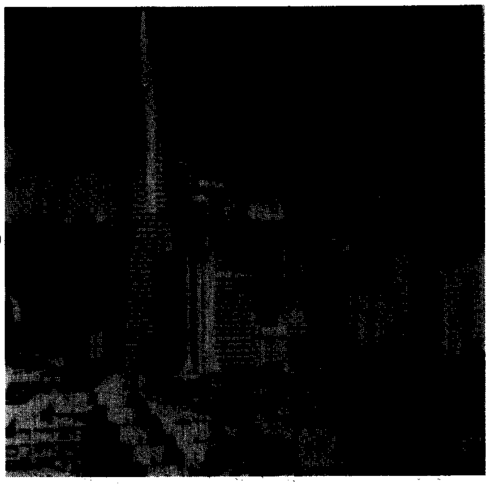

(c)

Fig. 11. (a) Output image produced by $\beta$-continuation network. Here, the parameter values are $\alpha = 1 \times 10^{-3}$, $\gamma = 1 \times 10^{-5}$, $\lambda_f = 3 \times 10^{-5}$, and $\beta = 0$. (b) Output image produced by $\beta$-continuation network. Here, the parameter values are $\alpha = 1 \times 10^{-3}$, $\gamma = 1 \times 10^{-5}$, $\lambda_f = 3 \times 10^{-5}$, and $\beta = 2 \times 10^{4}$. (c) Output image produced by $\beta$-continuation network. Here, the parameter values are $\alpha = 1 \times 10^{-3}$, $\gamma = 1 \times 10^{-5}$, $\lambda_f = 3 \times 10^{-5}$, and $\beta = 1 \times 10^{6}$.


(a)


(b)

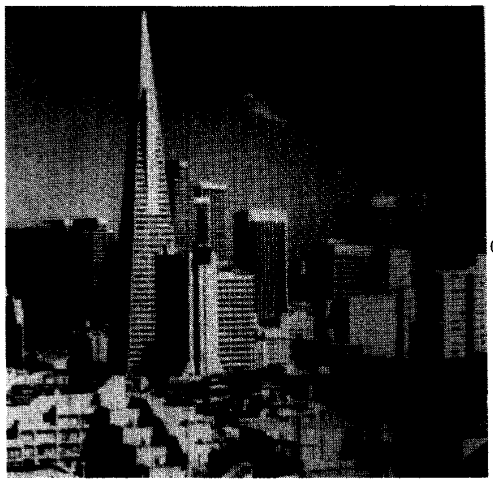

(c)

Fig. 12. (a) Output image produced by $\lambda_f$-continuation network. Here, the parameter values are $\alpha = 1 \times 10^{-3}$, $\gamma = 1 \times 10^{-5}$, $\beta = 1 \times 10^{6}$, and $\lambda_f = 1$. (b) Output image produced by $\lambda_f$-continuation network. Here, the parameter values are $\alpha = 1 \times 10^{-3}$, $\gamma = 1 \times 10^{-5}$, $\beta = 1 \times 10^{6}$, and $\lambda_f = 1 \times 10^{-3}$. (c) Output image produced by $\lambda_f$-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 that produced the results of Fig. 11, but that the output image is much closer to the input image shown in Fig. 10.

value of $\lambda_f$. The fixed parameter values were $\alpha = 1 \times 10^{-3}$, $\gamma = 1 \times 10^{-5}$, and $\beta = 1 \times 10^{6}$; the value of the $\lambda_f$ was linearly varied as a function of time from $\lambda_f = 1$ to $3 \times 10^{-5}$. Fig. 12(a) shows the output of the network at the beginning of the continuation when $\lambda_f = 1$; Fig. 12(b) shows the output of the network at an intermediate point of the continuation when $\lambda_f = 1 \times 10^{-3}$; Fig. 12(c) shows the output of the network at the end of the continuation when $\lambda_f = 3 \times 10^{-5}$. Note that the final parameter values of this network are identical to those for the network that produced the results of Fig. 11.

### *5.5. 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 min and 29 min, 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.

## VI. Conclusion

In this paper we presented the algorithms in CMVSIM, a program for performing the transient simulation of grid-based analog signal processors on a massively parallel computer. In particular, we showed that by using a grid-based equation formulation approach, and a block-diagonal preconditioned CGS algorithm, CMVSIM is able to efficiently exploit massive parallelism to simulate a very general class of grid-based signal processors. Experimental results presented demonstrate that CMVSIM running on a full-size CM can provide a 650 times speedup over, what is to the authors' knowledge, the *best serial algorithm* running on a SUN-4/490 workstation. Even with only the 1/4 size CM-2 available to the authors, it was possible to simulate a 128 × 128 example of Mead's Silicon Retina, a circuit with over 300 000 SPICE level 3 MOS devices, in about half an hour. The same circuit simulated with what our investigation suggested was the best serial algorithm would take an estimated three SUN-4/490 CPU days—assuming the workstation had enough memory to accommodate the problem.

Future work is focused on extending the simulator to allow more general circuits. For example, CMVSIM currently only supports Manhattan-style circuit grids, though more general structures, like hexagonal grids, can be incorporated in a straight-forward manner. Also, CMVSIM can be enhanced to allow the simulation of circuits which are "mostly regular," like dynamic memories, by incorporating the ability to run in a hybrid mode. For the dynamic memory example, this would mean that the regular portion of the circuit, the storage array and interconnect, would be simulated on the CM, but the less regular peripheral circuitry, like sense amplifiers and row and column drivers, would be simulated serially on the front-end. Newer parallel machines, such as the CM-5, should allow such hybrid simulations to be performed without resorting to the front-end machine.

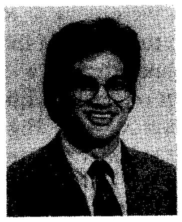
## Acknowledgments

The authors are grateful to Thinking Machines Corporation, in particular Rolf Fiebrich, James Frankel, Karen Jourdenais, and Josh Simons, for providing hardware and software support for the development of the simulator. The authors would also like to thank Prof. John Wyatt and the MIT Vision Chip Project group for providing the motivation for this work. Finally, the authors would like to thank the reviewers for their suggestions on how to improve the presentation of this work.

## 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, MIT, Oct. 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 *Proc. Int. Symp. on Circuits and Systems*, New Orleans, LA, May 1990, pp. 987-991.
[5] A. Lumsdaine, "Theoretical and Practical Aspects of Parallel Numerical Algorithms for Initial Value Problems, with Applications", Ph.D. dissertation, MIT, Cambridge, MA, 1992.
[6] L. W. Nagel, "SPICE2: A computer program to simulate semiconductor circuits," Tech. Rep. ERL M520, Electronics Res. Lab. Rep., Univ. of Calif., Berkeley, May 1975.
[7] A. Lumsdaine, M. Silveira, and J. White, "CMVSIM users' guide," Res. Lab. of Electronics, MIT, 1990.
[8] K. S. Kundert, tech. rep. Private Notes.
[9] A. J. Hoffman, M. S. Martin, and D. J. Rose, "Complexity bounds for regular finite difference and finite element grids," *SIAM J. Numer. Anal.*, vol. 10, pp. 364-369, 1973.
[10] P. Worley and R. Schreiber, "Nested dissection on a mesh-connected processor array," in *New Computing Environments: Parallel, Vector and Systolic* (A. Wouk, ed)., pp. 8-38. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1986.
[11] H. C. Elman, "Iterative Methods for Large Sparse Nonsymmetric Systems of Linear Equations," Ph.D. dissertation. Comp. Sci. Dept., Yale Univ., New Haven, CT, 1982.
[12] L. M. Silveira, "Circuit simulation algorithms for massively parallel processors," Master's thesis, MIT, May 1990.
[13] P. Sonneveld, "CGS, a fast Lanczos-type solver for nonsymmetric linear systems," *SIAM J. Sci. Statist. Comput.*, vol. 10, pp. 36-52, 1989.
[14] 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 *Proc. Int. Conf. on Computer Aided-Design*, Santa Clara, CA, Nov. 1989, pp. 408-411.
[15] 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 *Proc. Int. Conf. on Computer Aided-Design*, Santa Clara, CA, Nov. 1990, pp. 446-449.

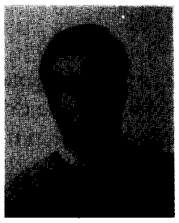
[16] K. S. Kundert and A. L. Sangiovanni-Vincentelli, "Sparse user's guide," Tech. Rep., Dept. of Elect. Eng. and Comp. Sci., Univ. of Calif., Berkeley, 1988.
[17] T. Quarles, "Analysis of Performance and Convergence Issues for Circuit Simulation," Ph.D. dissertation, Univ. of Calif., Berkeley. Apr. 1989. Available as UCB/ERL M89/42.
[18] J. A. Meijerink and H. A. van der Vorst, "An iterative solution method for linear systems of which the coefficient matrix is a symmetric *M*-matrix," *Math. Comp.*, vol. 31, pp. 148-162, 1977.
[19] D. M. Webber and A. Sangiovanni-Vincentelli, "Circuit simulation on the Connection Machine," in *Proc. 24th ACM/IEEE Design Automation Conf.*, June 1987.
[20] A. Lumsdaine, M. Silveira, and J. White, "SIMLAB programmer's guide," Res. Lab. of Electronics, MIT. Unpublished, 1990.
[21] M. Silveira, A. Lumsdaine, and J. White, "SIMLAB users' guide," Res. Lab. of Electronics, MIT, 1990.
[22] "Connection Machine Model CM-2 Technical Summary," Tech. Rep.-General TR 89-1, Thinking Machines Corp., Cambridge, MA, May 1989.
[23] A. Lumsdaine, M. Silveira, and J. White, "CMVSIM programmer's guide," Res. Lab. of Electronics, MIT. Unpublished, 1990.
[24] *C* Programmer's Guide*. Cambridge, MA: Thinking Machines Corporation, Nov. 1990.
[25] *C* User's Guide*. Cambridge, MA: Thinking Machines Corporation, Nov. 1990.
[26] A. Lumsdaine, J. Wyatt, and I. Elfadel, "Nonlinear analog networks for image smoothing and segmentation," *J. VLSI Signal Processing*, vol. 3, pp. 53-68, 1991.



**Andrew Lumsdaine** (S'84-M'92) received the SBEE, SMEE, and Ph.D. degrees from MIT in 1984, 1986, and 1992, respectively.

During 1986, he worked as an engineer in the manufacturing development group at the Packard Electric Division of General Motors. He is currently on the faculty in the Department of Computer Science and Engineering at the University of Notre Dame. His research interests include parallel processing, scientific computing, numerical methods, and VLSI circuit and semiconductor device simulation.

Dr. Lumsdaine is a member of SIAM, Sigma Xi, and Eta Kappa Nu.



**Luís Miguel Silveira** (S'85) received the Engineer's (summa cum laude) and Master's degrees in electrical and computer engineering in 1986 and 1989, from Instituto Superior Técnico at the Technical University of Lisbon, and the M.S. and E.E. degrees in 1990 and 1991 from the Massachusetts Institute of Technology, Cambridge, MA. He is currently working on his Ph.D. in Electrical Engineering and Computer Science at MIT.

His research interests are in various aspects of computer-aided design of integrated circuits with emphasis on numerical simulation methods.

Mr. Silveira is a member of Sigma Xi.

**Jacob White** (A'88), for a photograph and a biography, please see page 823 of the June, 1993 issue of this TRANSACTIONS.