Massively Parallel Simulation Algorithms for Grid-Based Analog Signal Processors

Andrew Lumsdaine L. Miguel Silveira Jacob K. White
Research Laboratory of Electronics
Department of Electrical Engineering and Computer Science
Massachusetts Institute of Technology
Cambridge, MA 02139

October 17, 1995

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.

1 Introduction

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

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

In this paper, algorithms are presented for simulating grid-based analog signal processors on a massively parallel computer. In Section 2, motivation is provided for this work by way of an

---

idealized example of a grid-based analog signal processor, and a general model of grid-based circuits is developed. The simulation algorithms for performing transient simulation of grid-based circuits are discussed in Section 3 and the massively parallel implementation of the algorithms is presented in Section 4. Experimental results using the Connection Machine are presented in Section 5. Finally, conclusions and suggestions for further work are contained in Section 6.

2 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 4 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 simulation program CMVSIM, by briefly describing the program's input files.

2.1 Motivational Problem

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

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

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

In a physical implementation of the image smoothing and segmentation circuit, the idealized elements in the circuit in Figure 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 5).

2.2 General Array Description

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

In order to create an $N \times N$ grid circuit of subcircuits, a single subcircuit must be replicated N^2 times, and then each subcircuit must be connected to its four nearest neighbors. The following

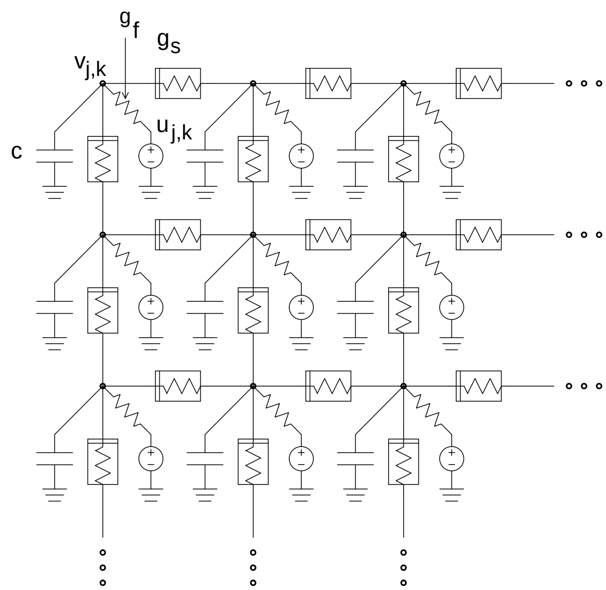

Figure 1: Grid of nonlinear resistors.

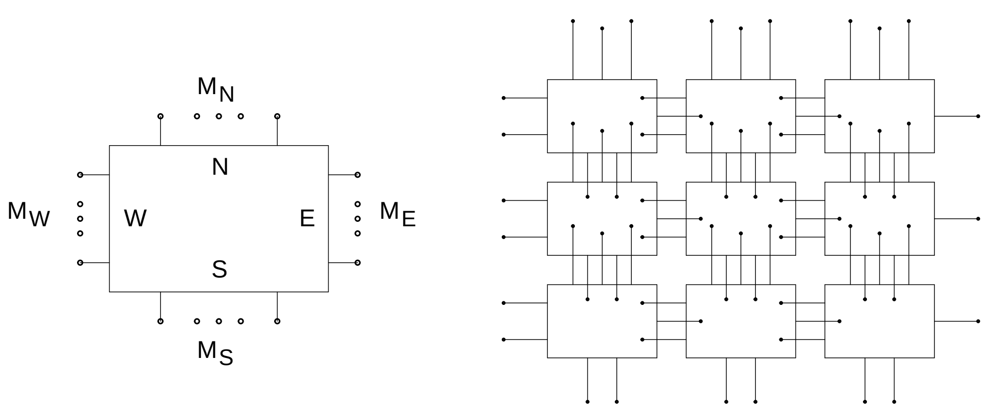

Figure 2: A single subcircuit, shown here as a multiterminal element, and a grid constructed of such elements by attaching nodes of each element to nodes of its neighbors.

proposition and its corollaries provide a means of more formally describing the grid circuit behavior, given the description of the behavior of the individual subcircuits.

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

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

Proof. Define $\boldsymbol{H} : \mathbf{R}^n \to \mathbf{R}^{n+m}$ 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 $\boldsymbol{H}\boldsymbol{v}(t)$ for $\tilde{\boldsymbol{v}}(t)$ 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 $\boldsymbol{q}(\boldsymbol{v}(t),t)$, 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, $q_j = \sum_{k \in K_j} \tilde{q}_k + \tilde{q}_j$, and therefore, $\boldsymbol{q}(\boldsymbol{v}(t),t) = \boldsymbol{H}^T \tilde{\boldsymbol{q}}(\boldsymbol{H}\boldsymbol{v}(t),t)$. By an analogous argument, if follows that $\boldsymbol{i}(\boldsymbol{v}(t),t) = \boldsymbol{H}^T \tilde{\boldsymbol{i}}(\boldsymbol{H}\boldsymbol{v}(t),t)$. ☐

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

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

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

Remark. The matrix $\boldsymbol{H}$ 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 $\{k_1,\ldots,k_m\} \subset \{1,\ldots,n+m\}$ to the n nodes in $\{1,\ldots,n+m\} \backslash \{k_1,\ldots,k_m\}$, and then renumbering the nodes in $\{1,\ldots,n+m\} \backslash \{k_1,\ldots,k_m\}$ from from 1 to n. This node renumbering implies that the generalized $\boldsymbol{H}$ matrix typically will *not* have the diagonal submatrix structure of the $\boldsymbol{H}$ 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

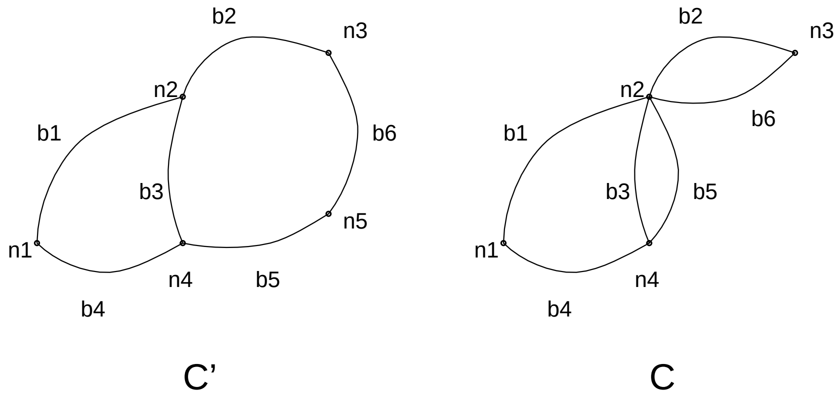

Figure 3: Example graphs for circuits $\tilde{\mathcal{C}}$ and $\mathcal{C}$. The circuits both have branches b1 through b6; circuit $\tilde{\mathcal{C}}$ has nodes n1 through n5, while circuit $\mathcal{C}$ has nodes n1 through n4. The topological matrix relating the nodes for this example is given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

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 two or more nodes in the decomposition, one of the nodes will also be an internal node, while the others are called *terminal* nodes. In general, one can designate any one of the separated nodes to be internal. However, since we will later see that there is a communication pattern imposed by the internal/terminal node designation, we adopt the convention that the internal node is the one that is a nearest neighbor to the others. Such a node will always be found because we are separating the grid circuit nearest-neighbor fashion. We formalize this approach to equation formulation in the following corrollary.

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

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

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

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

Remark. If all N^2 subcircuits are identical, then $\tilde{n} = N^2 M_{int} + N^2 M_{term}$ and $n = N^2 M_{int}$. However, 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 $\boldsymbol{J}_q = \frac{\partial \boldsymbol{q}}{\partial \boldsymbol{v}}$, $\boldsymbol{J}_i = \frac{\partial \boldsymbol{i}}{\partial \boldsymbol{v}}$, $\boldsymbol{J}_{\tilde{q}} = \frac{\partial \tilde{\boldsymbol{q}}}{\partial \boldsymbol{v}}$, and $\boldsymbol{J}_{\tilde{i}} = \frac{\partial \tilde{\boldsymbol{i}}}{\partial \boldsymbol{v}}$ be the Jacobian matrices for the functions $\boldsymbol{q}$, $\boldsymbol{i}$, $\tilde{\boldsymbol{q}}$, and $\tilde{\boldsymbol{i}}$ described in Corollary 2.2, respectively, and let $\boldsymbol{H}$ be the topological matrix relating $\boldsymbol{q}$ to $\tilde{\boldsymbol{q}}$ and $\boldsymbol{i}$ to $\tilde{\boldsymbol{i}}$. Then, $\boldsymbol{J}_q$ is related to $\boldsymbol{J}_{\tilde{q}}$ and $\boldsymbol{J}_i$ is related to $\boldsymbol{J}_{\tilde{i}}$ by

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

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

Proof. The result follows directly from 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 Figure 1 is shown in Figure 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 Figure 4 in the plane with 90 degree increments. 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

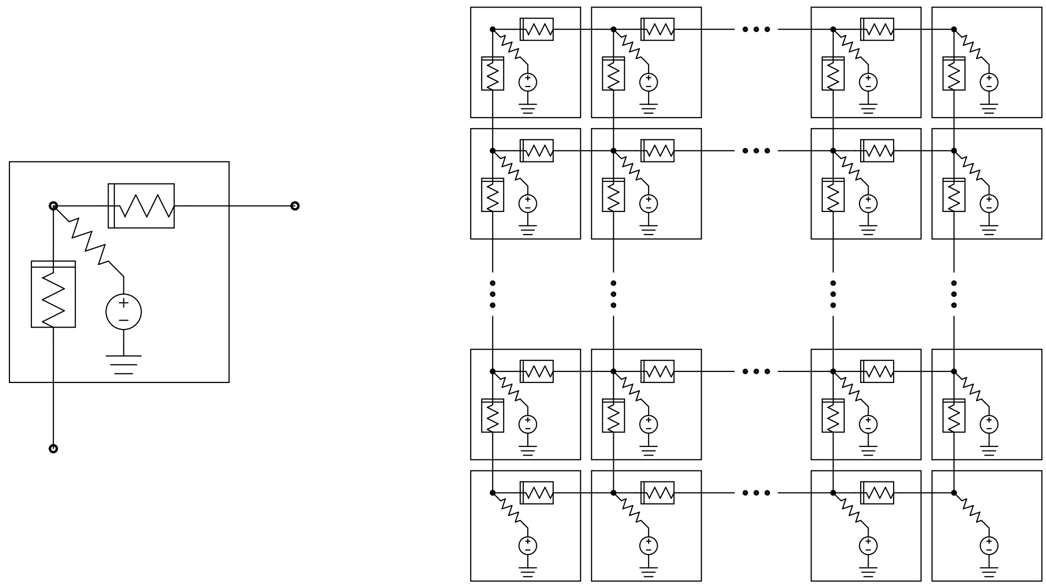

Figure 4: Decomposition of the example circuit shown in Figure 1. Note that the subcircuits on the east and south borders of the grid differ from the subcircuits elsewhere in the grid.

lres.rel.e

```
global 0

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

lres.rel.s

```
global 0

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

lres.rel.h

```
global 0

; common here

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

are common to all files. This is done with a "common" comment in the `here` circuit file. The "common" comment has the form:

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

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.

As a complete example, the contents of the circuit files to specify a resistive grid are shown in the files `lres.rel.e`, `lres.rel.s`, and `lres.rel.h`. Refer to the "CMVSIM users' guide" [17] for more details on using CMVSIM.

3 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

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

Here, $\boldsymbol{v}(t), \boldsymbol{q}\left(\boldsymbol{v}(t), t\right), \boldsymbol{i}\left(\boldsymbol{v}(t), t\right) \in \mathbf{R}^n$ 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:

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

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

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

where

$$\begin{aligned}&\boldsymbol{F}\left(\boldsymbol{v}(t+h), t+h\right) = \\ &\quad \frac{2}{h}\left[\boldsymbol{q}\left(\boldsymbol{v}(t+h), t+h\right) - \boldsymbol{q}\left(\boldsymbol{v}(t), t\right)\right] + \left[\boldsymbol{i}\left(\boldsymbol{v}(t+h), t+h\right) + \boldsymbol{i}\left(\boldsymbol{v}(t), t\right)\right]\end{aligned} \tag{7}$$

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

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

In "classical" circuit simulators such as SPICE [6], the linear system of equations for each Newton iteration is solved by some form of sparse Gaussian elimination. Sparse Gaussian elimination is not well-suited to the present problem for several reasons. First, because of the structure of the matrix generated by grid-type circuits, sparse Gaussian elimination creates substantially more fill than for typical circuits with an equivalent number of nodes. Second, it is well known that sparse Gaussian elimination is difficult to parallelize. Therefore, a parallelizable iterative method is sought, such as a conjugate-direction method [7].

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 [7, 8]. The present discussion will be restricted to CGS, presented in Algorithm 3.1, since experience has shown that it is the most efficient of the methods examined (see also [9, 10]).

One reason for the suitability of conjugate direction methods to the problem under consideration is that the convergence rates of these methods depend on the condition number of $\boldsymbol{J}_F$. By taking timesteps small enough, the capacitive portion of the Jacobian, which tends to be better conditioned than the conductive portion, will dominate (see [11]).

To demonstrate the effectiveness of the conjugate-direction methods, in Table 1 the CPU time required to compute the transient analysis of the network in Figure 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 5. The conjugate-direction methods shown in Table 1 are conjugate gradient, conjugate gradient squared (CGS), incomplete LU preconditioned

Algorithm 3.1 (Conjugate Gradient Squared algorithm for solving $\boldsymbol{Ax} = \boldsymbol{b}$).

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

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

Table 1: Comparison of serial execution time for the transient simulation of the circuit in Figure 1, when using direct, and conjugate-direction linear system solvers. For this example, $g_f = 3.0 \times 10^{-5}\Omega^{-1}$ and g_s has a conductance of $1.0 \times 10^{-3}\Omega^{-1}$ when linearized about zero. 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.

CG (ILUCGS), 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. For these examples, the complexity of a single linear system solution with Gaussian elimination is approximately $\mathcal{O}(N^{1.7})$ (for N circuit nodes); the corresponding complexity of ILUCG is approximately $\mathcal{O}(1.25)$. As the grid becomes larger, the number of timesteps and Newton iterations also increases. Hence, the total linear solution time appears to have a higher complexity as a function of circuit size than does the single linear solution time.

As the table indicates, sparse Gaussian elimination is much slower than the conjugate-direction methods, especially for larger problem sizes. Note that this example problem has a symmetric Jacobian, and therefore the more efficient CG algorithm can be used instead of CGS. For arrays constructed of semiconductor transistors, to be described in the following sections, the Jacobian will not be symmetric, and only CGS may be used.

4 Implementation

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

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

The general circuit simulation problem violates all three of the above constraints, and previous attempts at using a SIMD machine to accelerate circuit simulation have yielded only limited success [11, 12]. 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 2. The problem is mapped onto an $N \times N$ processor array by assigning the data for one subcircuit to each processor (see Figure 5).

The characteristics of the grid circuit can be obtained with this mapping by using Corollary 2.2, as described in Sections 4.2 and 4.3. Note that $\tilde{\boldsymbol{v}}$, $\tilde{\boldsymbol{q}}(\tilde{\boldsymbol{v}})$, and $\tilde{\boldsymbol{i}}(\tilde{\boldsymbol{v}})$ can be completely represented by keeping $(M_{int} + M_{term})$ components of these vectors on each processor. However, representing $\boldsymbol{v}$, $\boldsymbol{q}(\boldsymbol{v})$, and $\boldsymbol{i}(\boldsymbol{v})$ requires only M_{int} components of each vector on each processor. Therefore, the convention is adopted that the components of $\tilde{\boldsymbol{v}}$, $\tilde{\boldsymbol{q}}(\tilde{\boldsymbol{v}})$, and $\tilde{\boldsymbol{i}}(\tilde{\boldsymbol{v}})$ which correspond to the internal nodes are stored in locations $\{1, \ldots, M_{int}\}$ and the components which correspond to terminal nodes are stored in locations $\{M_{int}+1, \ldots, M_{int}+M_{term}\}$. With this convention, the vectors $\boldsymbol{v}$, $\boldsymbol{q}(\boldsymbol{v})$, and $\boldsymbol{i}(\boldsymbol{v})$ are simply the first M_{int} locations of the vectors $\tilde{\boldsymbol{v}}$, $\tilde{\boldsymbol{q}}(\tilde{\boldsymbol{v}})$, and $\tilde{\boldsymbol{i}}(\tilde{\boldsymbol{v}})$, respectively.

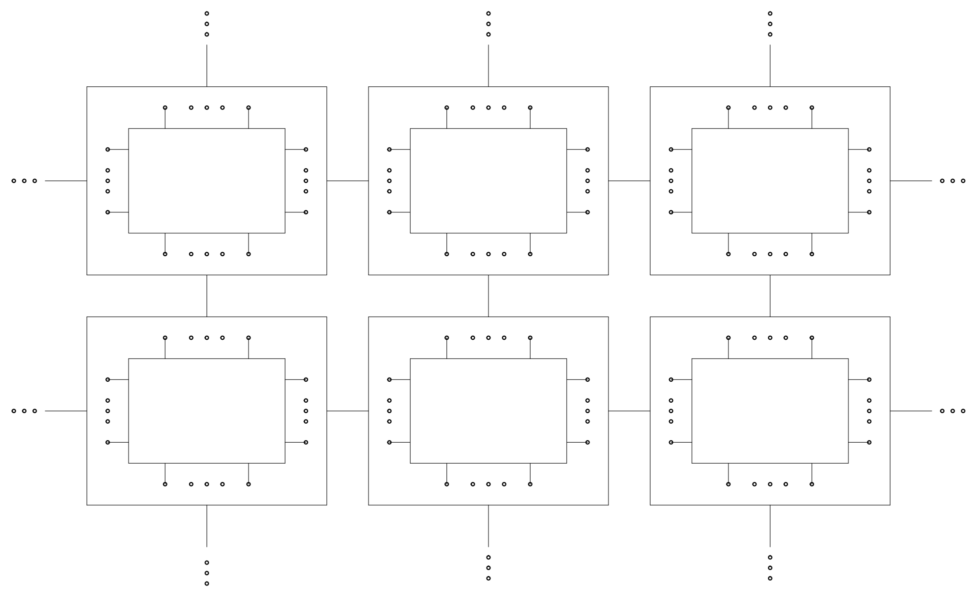

Figure 5: Mapping of subcircuits to processor grid. Each processor, represented by a single box, contains the data necessary for simulating a single subcircuit. The lines connecting the boxes represent the inter-processor communication network.

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

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

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

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

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

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

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 $\boldsymbol{y} = \boldsymbol{J}_F \boldsymbol{x}$ is computed according to $\boldsymbol{y} = \boldsymbol{H}^T \boldsymbol{J}_{\tilde{F}} \boldsymbol{H} \boldsymbol{x}$, using the result of Corollary 2.3, with the following sequence of operations:

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

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

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

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

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 $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ directly, the conjugate-direction method is applied to the equivalent system

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

Typically, $\boldsymbol{P}$ is chosen so that the system $\boldsymbol{P}\boldsymbol{z} = \boldsymbol{r}$ is easy to solve for $\boldsymbol{z}$ 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 Figure 2, the structure of the Jacobian matrix $\boldsymbol{J}_F$ is that of a block diagonal matrix with block off-diagonal bands. The diagonal blocks are of size $M_{int} \times M_{int}$; the block off-diagonal bands are of size $M_N \times M_N$, $M_E \times M_E$, $M_W \times M_W$, and $M_S \times M_S$. This suggests a block iterative method for solving (6), using the diagonal blocks of $\boldsymbol{J}_F$ as the preconditioner. A block iterative scheme is particularly well suited to a SIMD implementation, since each block can be solved simultaneously in parallel. Note that although $\boldsymbol{J}_{\tilde{F}}$ is already block diagonal, inverting its blocks is not the same as inverting the block diagonal portion of $\boldsymbol{J}_F$.

Rather, let $\boldsymbol{J}_F = \boldsymbol{P} + \boldsymbol{R}$, where $\boldsymbol{P}$ is the block diagonal part of $\boldsymbol{J}_F$. To use $\boldsymbol{P}$ as a preconditioner in the CGS algorithm, it must be formed from $\boldsymbol{J}_{\tilde{F}}$. Let $\tilde{R}_{j,k}$ be the part of $\boldsymbol{J}_{\tilde{F}}$ corresponding to the coupling between internal and terminal nodes of the same subcircuit — this coupling will become the off-diagonal coupling blocks in $\boldsymbol{J}_F$. Define $\tilde{\boldsymbol{P}}$ by $\boldsymbol{J}_{\tilde{F}} = \tilde{\boldsymbol{P}} + \tilde{\boldsymbol{R}}$ (see Figure 6). Using Corollary 2.3, note that

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

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

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

2. Solve $\boldsymbol{P}\boldsymbol{z} = \boldsymbol{r}$.

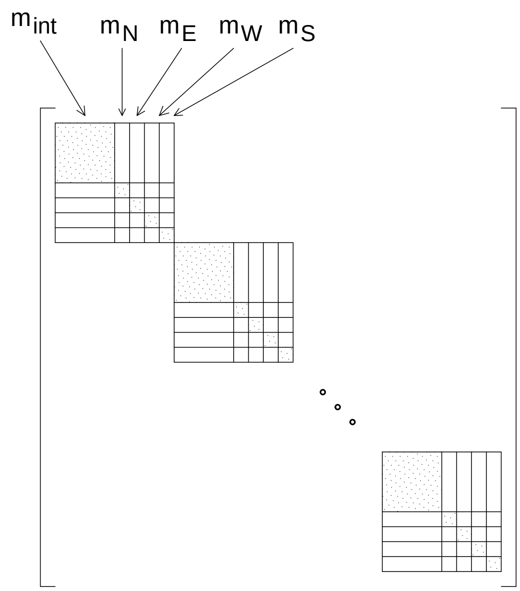

Figure 6: Definition of $\tilde{\boldsymbol{P}}$. Since the coupling between internal nodes and terminal nodes will become off-diagonal blocks in $\boldsymbol{J}_F$, these coupling elements are removed from $\boldsymbol{J}_{\tilde{F}}$ to form $\tilde{\boldsymbol{P}}$.

5 Experimental Results

In this section, experimental results are presented that compare serial and parallel execution times for several types of grid-based analog signal processors. To make the results as meaningful as possible, the fastest serial algorithm is compared with the fastest parallel algorithm. The programs used to test the serial and parallel programs both use the SIMLAB program — a SPICE-like circuit simulator developed at MIT [13, 14] — 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) [15]. The bit-serial processors are clustered together into groups of 16 within a single integrated circuit, and these IC's are connected together in a 12-dimensional hypercube. Two IC's, or 32 processors, share a single Weitek FPU. Note that a fully configured CM-2 contains 2048 times as much floating point hardware as a conventional computer containing a single Weitek FPU (e.g., a SUN-4).

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

The serial experiments were run using the VSIM (**V**ision **SIM**ulation) program. The VSIM program is essentially SIMLAB with idealized nonlinear elements and image I/O support added. The parallel experiments were run using the CMVSIM (**C**onnection **M**achine **V**ision **SIM**ulation) program [16, 17]. 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` [18, 19]. 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

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

Table 2: Tolerances used in simulations reported in Section 5

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 2. 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.

The larger grid problems could not be accomodated in the serial machine because of memory limitations, so it was necessary to estimate some of the serial CPU times. The estimates were produced by estimating the complexity of the linear solution time as a function of the growth of the problem size. This estimated function for the complexity was used to calculate estimated times for the larger grid sizes. Also, the largest grid size (256×256) did not admit a one subcircuit to one physical processor mapping on the 1/4 size CM-2 available to the authors. Rather than confuse issues by using the virtual processor mechanism of the CM to simulate the 256×256 example, the parallel execution times for the 256×256 circuit grids were also estimated. The fact that estimates were required at all highlights another advantage of using the CM — a large number of the problems studied in this section simply could not be simulated on the serial machine at all. This fact is as important as the serial/parallel execution time comparisons.

Linear Resistive Grid

Table 3 shows the results obtained while simulating the linear resistive circuit grid shown in Figure 7. The circuit for which the results are shown had a $1k\Omega$ resistance between neighboring nodes, a $33.33k\Omega$ resistance to ground, and a parasitic capacitance of $1\times10^{-15}F$ at each node. A random image was introduced to the network and the startup transient of the first 1×10^{-5} seconds was simulated.

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

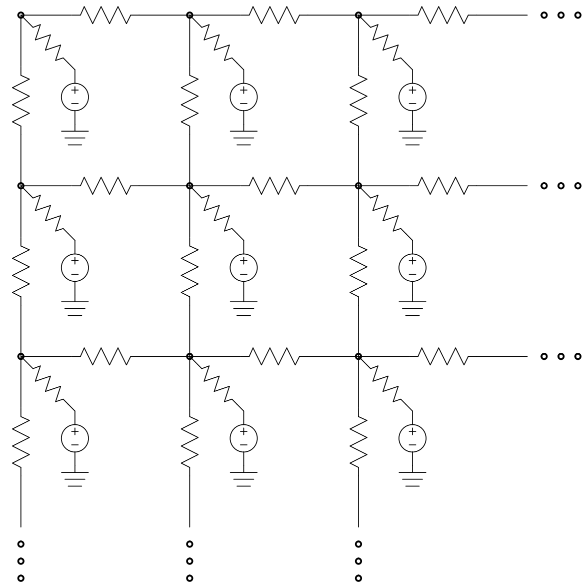

Figure 7: Linear resistive grid.

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

Table 3: Experimental result: Linear circuit grid.

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

Table 4: Experimental result: Nonlinear circuit grid.

Nonlinear Resistive Grid

Table 4 shows the results obtained while simulating the nonlinear resistive circuit grid shown in Figure 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 $10k\Omega$ to ground, a parasitic capacitance of $1\times10^{-7}F$ and parameter values for the nonlinear resistance of $\alpha = 1\times10^{-3}$, $\gamma = 2\times10^{-5}$ with β being swept from 0 to 1×10^5 over the simulation interval. A random image was introduced to the network, the DC solution was found with $\beta = 0$ (i.e., so that the network was linear), and then a transient simulation of one second was performed, during which β was increased from 0 to 1×10^5. The reasons for this type of simulation are explained in detail in [20].

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

Mead's Silicon Retina

Tables 5 and 6 show the results obtained while simulating Mead's Silicon Retina as described in [1] and shown in Figure 8. Table 5 shows results obtained when presenting the circuit with a constant image while Table 6 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 reduced the CPU time by up to a factor of four over plain CGS.

For the 128×128 grid, the CM achieved a speedup of about a factor of 95 for the circuit with the constant input image and a speedup of over a factor of 144 for the circuit with the random input image. If the serial and parallel results are extrapolated to a 256×256 grid, the CM should be able to achieve a speedup of about a factor of 400 for the circuit with the constant input image and a speedup of over a factor of 650 for the circuit with the random input image.

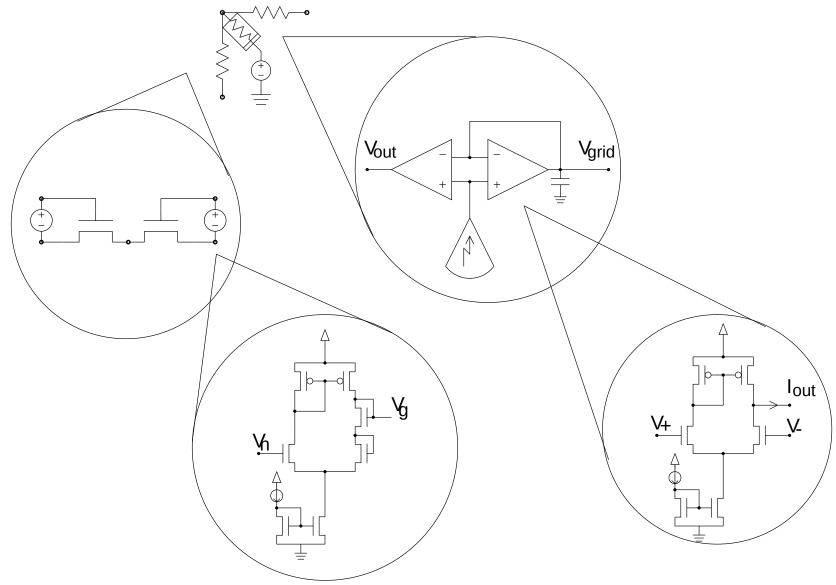

Figure 8: Mead's Silicon Retina. The simulated chip contains 23 MOS level 3 devices per cell for a total of 376,320 devices and 261,888 nodes in a 128×128 grid.

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

Table 5: Experimental result: Mead's Silicon Retina with constant input image.

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

Table 6: Experimental result: Mead's Silicon Retina with random input image.

Figure 9: 256×256 image of the San Francisco sky line.

Processing Images

One of the most significant features of CMVSIM is that the program can be used to investigate how particular vision circuits process images. To demonstrate this use of CMVSIM, the results of two experiments are presented where the same network is subjected to two different types of continuations, resulting in drastically different output images (see [20] for detailed treatment of the continuations). The network used is shown in Figure 1 and uses the nonlinear element described by (10); let the linear conductance to ground be called λ_f. Figure 9 shows a 256×256 image — a portion of the San Francisco sky line — used as the input image.

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

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

Figure 9(a): Output image produced by β-continuation network. Here, the parameter values are $\alpha = 1\times10^{-3}$, $\gamma = 1\times10^{-5}$, $\lambda_f = 3\times10^{-5}$, and $\beta = 0$.

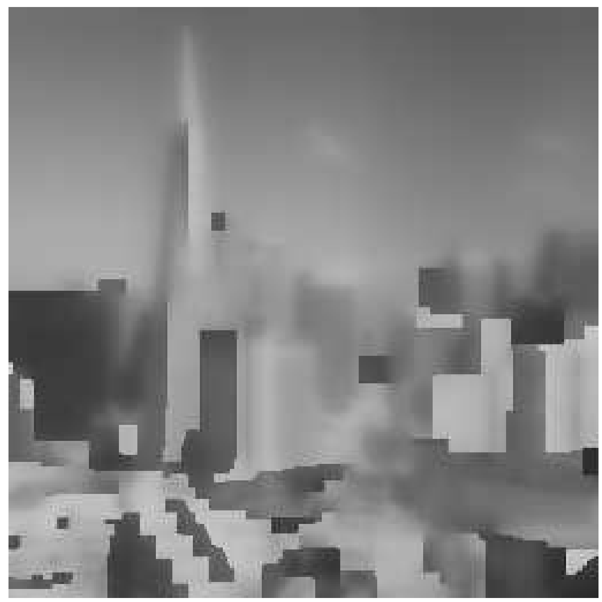

Figure 9(b): Output image produced by β-continuation network. Here, the parameter values are $\alpha = 1\times10^{-3}$, $\gamma = 1\times10^{-5}$, $\lambda_f = 3\times10^{-5}$, and $\beta = 2\times10^{4}$.

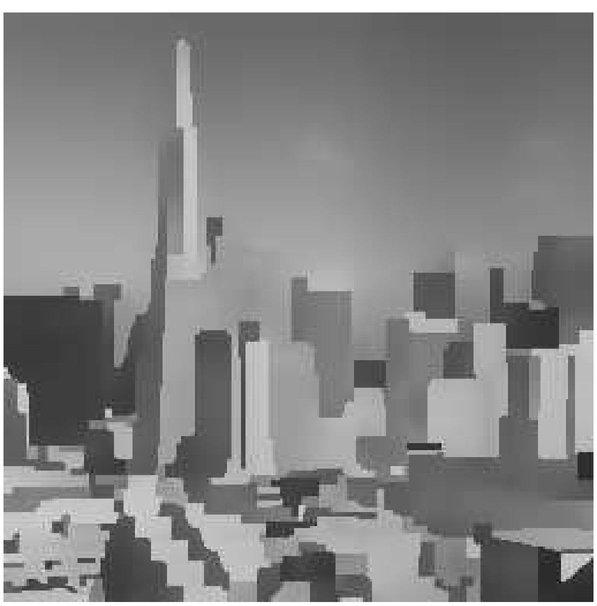

Figure 9(c): Output image produced by β-continuation network. Here, the parameter values are $\alpha = 1\times10^{-3}$, $\gamma = 1\times10^{-5}$, $\lambda_f = 3\times10^{-5}$, and $\beta = 1\times10^{6}$.

Figure 9(a): Output image produced by λ_f-continuation network. Here, the parameter values are $\alpha = 1 \times 10^{-3}$, $\gamma = 1 \times 10^{-5}$, $\beta = 1 \times 10^{6}$, and $\lambda_f = 1$.

Figure 9(b): Output image produced by λ_f-continuation network. Here, the parameter values are $\alpha = 1 \times 10^{-3}$, $\gamma = 1 \times 10^{-5}$, $\beta = 1 \times 10^{6}$, and $\lambda_f = 1 \times 10^{-3}$.

Figure 9(c): Output image produced by λ_f-continuation network. Here, the parameter values are $\alpha = 1 \times 10^{-3}$, $\gamma = 1 \times 10^{-5}$, $\beta = 1 \times 10^{6}$, and $\lambda_f = 3 \times 10^{-5}$. Note that the final parameter values of this network are identical to those for the network of Figure 5, but that the output image is much closer to the input image shown in Figure 9.

Discussion

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

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

6 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 Connection Machine 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 Sun4/490 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 Connection Machine, 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

This work was supported by the Defense Advanced Research Projects Agency under Contract No. N00014-87-K-825 and by the Portuguese INVOTAN committee. 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, Massachusetts Institute of Technology, October 1990.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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