A Survey of Third-Generation Simulation Techniques

GARY D. HACHTEL, FELLOW, IEEE, AND
ALBERTO L. SANGIOVANNI-VINCENTELLI, SENIOR MEMBER, IEEE

Invited Paper

Abstract—We present a review of recent work on circuit simulation techniques which are "third generation" in that they go beyond the Sparse Gauss Elimination, Newton Iteration, Stiff Implicit time integration approach which mark second-generation circuit simulators such as SPICE-II and ASTAP-II. Third generation simulators such as MOTIS, DIANA, and SPLICE have rejected one or more of these principal features in their quest for size and speed capabilities commensurate with the requirements of the VLSI era. We attempt to present a unified treatment of the various and disparate types of third generation simulators based on the concepts of large-scale decomposition theory. In particular we shall describe and classify simulators in terms of the role played by certain matrix forms in their formulation, namely Bordered Block Diagonal (BBD), Bordered Block Triangular (BBT), and Bordered Lower Triangular (BLT).

I. Introduction

CIRCUIT SIMULATION, which matured in the 1970's [1]-[3], has established itself as a significant design aid and a significant cost item as well, in most large integrated circuit (IC) design houses. This fact is reflected by the presence in such houses of large mainframe computers dedicated solely to circuit simulation.

The spectacular growth in the scale, measured in device count, of IC's being designed in the VLSI microelectronics era has started and intensified a search for "third generation" methods of circuit simulation. Similar developments have stimulated the same kind of research in other disciplines such as electrical power distribution, chemical engineering plants, logic design, and operations research. The common theme is that the systems being designed take too much cpu time and/or too much storage for economically realistic simulation support for the design effort. As a result, the search is on for alternatives to circuit simulators that were once considered revolutionary [1]-[3], but are now regarded as "standard." This paper is devoted to a tutorial review of "third-generation" simulators and simulation techniques [4]-[17], i.e., simulators and techniques which, in their quest for larger scale capabilities, have departed radically from the approach defined in [1]-[3].

A similarly motivated paper has recently appeared [18]. The present paper differs in two respects. First, De Man [18] dealt exclusively with IC simulators and emphasized functionality, whereas the present paper has a more interdisciplinary scope and emphasizes algorithmic techniques. Second, the present paper attempts a unified treatment of the various disparate simulator types based on the concept of decomposition of large-scale systems. Specifically we shall classify and describe the various "third-generation" simulators in terms of the role played by certain matrix forms in their formulation, namely (cf. Fig. 1)

Bordered Block Diagonal (BBD)
Bordered Block Triangular (BBT)
Bordered Lower Triangular (BLT)

Block Diagonal (BD)
Block Triangular (BT)
Lower Triangular (LT).

Note that the BD and BT forms are just BBD and BBT forms without a border, and that the LT form is just the BT form with unit block sizes. It is common, however, when referring to the BT form, to assume that the diagonal blocks are irreducible, that is, cannot by themselves be decomposed by row and column permutation into a BT subform.

We have chosen decomposition as a theme because the modules of a decomposed system may be treated individually in special ways. For example, if storage is limited, sequential treatment of the individual modules permits much larger systems to be simulated. Also the modular approach permits advantage to be taken of machines with parallel architectures. Further, the modular approach is well suited for the exploitation of "latency," [13], [15], i.e., avoiding the expense of simulating modules which are not active at a given point in the simulation.

The plan of the paper shall be as follows. First, the introduction shall be completed by a brief sketch of what we mean by "standard" circuit simulation. Then, we begin the technical discussion in Section II with a description and classification of decomposition methods for linear, nonlinear, and dynamic systems. We continue in Section III with a discussion of algorithms for identifying an appropriate topological decomposition of a large system (for those cases where the decomposition is not functional, i.e., specified in advance by a hierarchical input language).

In Section IV we describe the steps taken in some simulators to exploit a given topological decomposition. This section begins with a discussion of node-tearing versus branch-tearing [10], an issue which remains to be settled after a topological

Manuscript received June 5, 1981. This work was supported in part by the Defense Advanced Research Projects Agency under Grant N00039-K-0251 and in part by the Air Force Office of Scientific Research under Grant F4920-79C0178.

G. D. Hachtel is with the Department of Electrical Engineering, University of Colorado, Boulder, CO 80309.

A. Sangiovanni-Vincentelli is with the Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA

0018/9219/81/1000-1264$00.75

decomposition has been determined. Then various data structures and numerical solution techniques are described which form the basis of some important third generation simulators [7], [10], [11], [16], [17]. In Section V we treat "temporal" decomposition methods (sometimes called "relaxation methods" or "indirect" methods [19]), thereby reviewing the line of research started by the celebrated MOTIS simulator [4], and continued through the development of SPLICE and DIANA [5], [6].

In Section VI, we further discuss some of the techniques of temporal decomposition, specifically describing the related concepts of latency [10], [13], event scheduling [12], [14], [15], and mixed mode simulation [4c]-[7], [11].

We begin our discussion of standard simulation by assuming that the system being simulated is described by a set of differential-algebraic equations

$$f(\dot{x}, x, t) = 0 \tag{1}$$

A very broad class of physical systems can be described in the form (1). "Standard" simulation is characterized for our present purposes by inclusion of all of the following algorithmic techniques for solving (1):

1) replacement of $\dot{x}$ by an astable or stiffly-stable [20] or backward difference [21] formula which is a function of $x(t)$,
2) automatic control of the time step h and differentiation order K so as to insure accuracy of the solution,
3) solution of the resulting nonlinear system by a quadratically convergent Newton's method,
4) solution of the linear algebraic equations involved in each Newton step by sparse Gaussian elimination.

In the approach of [20], [21], the time derivative of the vector x is approximated at time point t_{n+1} by

$$\dot{x}_{n+1} \equiv \alpha_0 (x_{n+1} - \bar{x}(K, n))/h \tag{2}$$

where K is the approximation order of the differentiation formula, $h \equiv (t_{n+1} - t_n)$, x_{n+1} is the computed value of $x(t_{n+1})$, and $\bar{x}(K, n) \equiv \sum_{k=1}^{K} \alpha_k x_{n+1-k}/\alpha_0$. Note that in "standard" simulation, the same order K and time step h is used for every component of $\dot{x}$. The α_K are chosen so that the solution is a polynomial of degree n passing through the last $n+1$ time points.

With the substitution (2), equation (1) can be expressed in the form

$$g(x_{n+1}, \bar{x}(K, n), t_{n+1}) = 0 \tag{3}$$

which is a system of nonlinear algebraic equations which must be solved at every time step t_{n+1}, $n = 1, 2, \cdots$, for the updated vector x_{n+1}.

The truncation error is controlled in standard simulators by monitoring

$$E_{\text{Trunc}} \equiv (x^P(t_{n+1}) - x_{n+1})/h \tag{4}$$

where $x^P(t_{n+1})$ is the value predicted for x_{n+1} by passing a polynomial of order K through the most recent $K+1$ time points, up to and including x_n. Because of the stability properties of (2) the time step h is controlled by accuracy requirements rather than by the stability of the difference equations (2). This would not be the case if simpler "explicit" formulas such as the "forward Euler" formula were employed, which avoid the solution of (3) at each time step but require extremely small time steps if the system equations are "stiff" (i.e., the Jacobian matrix exhibits widely separated eigenvalues at the operating points of interest).

One of the prime motivations for the development of sparse matrix technology [22], [23] was to solve the linear equations which arose when (3) was solved by Newton's method.

According to the preceding algorithmic technique 2, the solution to (3) is obtained in standard simulators by using the Newton's algorithm

$$\begin{aligned}
&x_{n+1}^0 \leftarrow x_n \\
&\text{For } v = 0, 1, \cdots \\
&\quad \text{Begin} \\
&\quad \partial g/\partial x(x_{n+1}^v, \bar{x}(K, n), t_{n+1})\, \Delta x = -g(x_{n+1}^v, \bar{x}(K, n), t_{n+1}) \\
&\quad x_{n+1}^{v+1} \leftarrow x_{n+1}^v + \Delta x \\
&\quad \textit{if } \delta^{v+1} \equiv \| x_{n+1}^{v+1} - x_{n+1}^v \| < \epsilon \text{ Then STOP; Else} \\
&\quad \text{End.}
\end{aligned} \tag{5}$$

The important feature of this algorithm is its quadratic convergence [19]. Ignoring roundoff errors, if the vth iteration incurs a suitably small error, say, 10^{-4}, then the error at the $(v+1)$th iteration will be, approximately, the previous error squared, i.e., $\delta^{v+1} = 10^{-8}$.

This ensures that (3) will be solved accurately, which is prerequisite to realizing the stability properties of (2) which in turn is required in order for the time step to be controlled by engineering accuracy rather than by expensive stability requirements. We will return to this key point in Section V, when we discuss MOTIS [4], which departed radically from the standard wisdom of the preceding algorithmic techniques 1-3, yet succeeded by restricting the type of network which could be simulated.

II. System Decomposition

Nonstandard simulators can be meaningfully classified by the decomposition techniques employed. A particular technique can be functionally and/or topologically motivated, and can be applied to any of the three main levels of circuit simulation, namely the time level, nonlinear equation level, or the linear equation level. There are two principal points of departure from the "standard" simulation approach which may be taken at each of these three levels, namely, "tearing" decomposition and "temporal" decomposition. The techniques we shall classify as "tearing" aim to retain the convergence and stability properties of the "standard" approach. In contrast, the techniques classified as "temporal" are related to the so-called "relaxation" or "indirect" methods, and are characterized by completely different convergence and stability properties. In this section we discuss the two decomposition levels at the linear level, and briefly introduce them at the nonlinear equation level and time level.

A. Decomposition of Linear Systems

Assume that the system to be solved is of the form

$$Ax = b, \qquad x, b \in R^n. \tag{6}$$

When we take the tearing approach, we regard the task of solving (6) efficiently, i.e., quickly (by exploiting parallel processing or latency) or with small storage, as equivalent to finding permutation matrices P and Q such that the permuted system

$$\hat{A}\hat{x} = \hat{b} \tag{7a}$$

$$\hat{A} \equiv PAQ \qquad \hat{x} \equiv Qx \qquad \hat{b} \equiv Pb \tag{7b}$$

$$\hat{A} \equiv B + CR^T \tag{7c}$$

can be solved by optimally exploiting the block structure

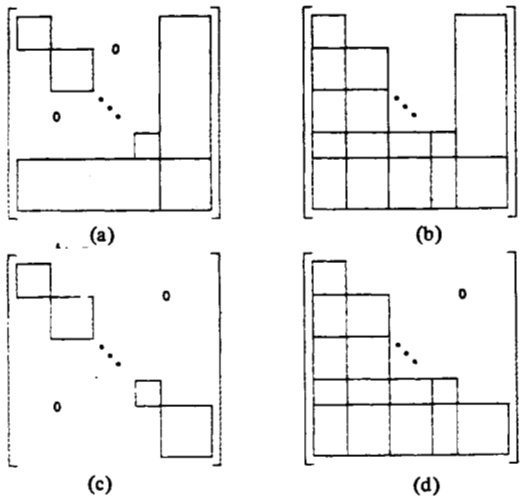


Fig. 1. Triangular matrix structures. (a) BBD. (b) BBT. (c) BD. (d) BT.

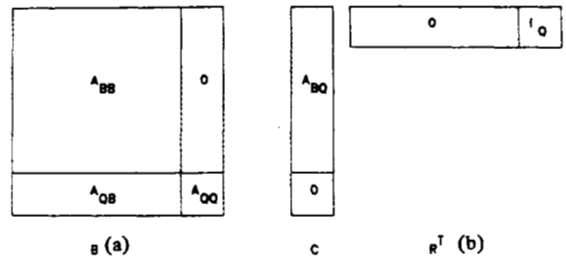


Fig. 2. Matrix decomposition.

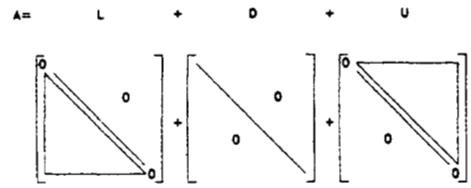


Fig. 3. Gauss-Seidel decomposition.

of the "torn" matrix B. That is, when the original matrix A is permuted into the form $\hat{A}$, which is BBD or BBT, the decomposition permits the reduced matrix B to assume the B_{BD} form of Fig. 1(c), or B_{BT} form of Fig. 1(d). The structure of the outer product CR^T is illustrated in Fig. 2(b). As we shall discuss in detail in Section III, the matrix CR^T can be thought of as "torn" from $\hat{A}$, leaving in its place the reduced matrix B. Tearing is formally equivalent to solving (7a) by applying the Sherman-Morrison-Woodbury formula [24],

$$\hat{x} = B^{-1}\hat{b} - B^{-1}(C(I_Q + R^T B^{-1} C)^{-1} R^T) B^{-1}\hat{b}. \tag{8}$$

It is not customary to solve (8) with explicit inverses. Instead, the procedure, or a variant of it, is usually employed. First, the block matrix B and an intermediate matrix Q are factorized, i.e.,

$$\begin{aligned} B &\leftarrow LU \\ Q &\equiv (I_Q + R^T U^{-1} L^{-1} C) \\ Q &\leftarrow L_Q U_Q. \end{aligned} \tag{9}$$

Once these LU factorizations are done, the solution is completed by the back substitutions

$$\begin{aligned} y &\leftarrow B^{-1}\hat{b} \equiv U^{-1}L^{-1}\hat{b} \\ z &\leftarrow Q^{-1}R^T y \equiv U_Q^{-1}L_Q^{-1}R^T y \\ \Delta y &\leftarrow U^{-1}L^{-1}Cz \\ x &\leftarrow y - \Delta y. \end{aligned} \tag{10}$$

Thus "tearing" the matrix C from A permits an initial approximation, i.e., y, to be computed by solving the system with the structured matrix B. At the expense of factoring a smaller matrix Q, and performing an extra back substitution with L and U, the solution x is completed by subtracting the correction term Δy from the approximation y.

With regard to efficiency we can make some general observations.

1) Since B has the structure of either B_{BD} or B_{BT}, $B^{-1} \equiv U^{-1}L^{-1}$ can be obtained blockwise. This is also true for the product of B^{-1} with various other terms in (8), which is the key point. This enables the block operations to be carried out concurrently to increase execution speed or serially to extend storage capability.

2) The rectangular matrix R, owing to its composition (Fig. 2(b)) as a zero matrix catenated with an identity matrix, operates solely as a selector, i.e., multiplication with this matrix requires no numerical operations.

3) Given that the LU factors of B are available (with the appropriate block structure of course), then various economies can be obtained by clever association of $B^{-1} \equiv U^{-1}L^{-1}$. For example, the quantity $B^{-1}C$ can be computed either as $U^{-1}(L^{-1}C)$ or $(U^{-1}L^{-1})C$. The most economical choice depends on the sparsity and block structure of the "tear" matrix C. George [25], [26], Hajj [27], and Guardabassi and Sangiovanni [50], have studied this association technique carefully and demonstrated the advantages which can be attained.

4) If the size of the border (number of columns in C) is small, the factorization and back substitutions of Q are cheap.

Various third-generation circuit simulation techniques based on this approach are discussed in Sections III and IV.

An alternative approach to solving (6) is based on the idea of "temporal" decomposition. The terms "relaxation methods" and "indirect methods" are sometimes used to describe the same process. In this approach, LU factorization is replaced by an iterative sweep through the equations of (7a), using the following procedure or a variant of it. Instead of obtaining the LU factorization of $\hat{A}$, we simply partition $\hat{A}$ into the form

$$\hat{A} \leftarrow \mathfrak{L} + \mathfrak{D} + \mathfrak{U} \tag{11}$$

where $\mathfrak{L}$ and $\mathfrak{U}$ are strictly (i.e., with zero diagonal) lower and strictly upper triangular matrices and $\mathfrak{D}$ is a diagonal matrix, as illustrated in Fig. 3. Then (7a) is solved by iteratively calling

Procedure Gauss–Seidel sweep (GSS) [19]:

For $v = 1, 2, \ldots,$
  Begin
  For $i = 1, 2, \ldots, n$
    Begin
    $x_i^{v+1} = \mathfrak{D}_{ii}^{-1}(b - \mathfrak{L}_i x^{v+1} - \mathfrak{U}_i x^v)$
    End
  $\delta^{v+1} \equiv \| x^{v+1} - x^v \|$
  End. (12a)

Here $\mathfrak{L}_i$ and $\mathfrak{U}_i$ stand for the ith row of the triangular matrices $\mathfrak{L}$ and $\mathfrak{U}$, and v stands for the iteration counter. The iteration stops when $\delta^{v+1} \equiv \|x^{v+1} - x^v\|$ is suitably small. Since it is easily shown that

$$\delta^{v+1} \equiv \|(I + \mathfrak{D}^{-1}\mathfrak{L})\mathfrak{D}^{-1}\mathfrak{U}\|\delta^v \equiv \|M\|\delta^v. \qquad (12b)$$

it can be seen that this solution process has the following properties:

1) the iteration converges for any initial value of x if and only if all the eigenvalues of M have modulus strictly less than 1;
2) the iteration converges in one step if the rows and columns of A are permuted so that $\mathfrak{U}$ is identically zero, in this case P and Q exist such that C is identically zero in (7), so that (8) and (12) are equivalent;
3) speed of convergence is improved if A is permuted into nearly lower triangular form;
4) convergence depends, in general, on the numerical properties of $\mathfrak{L}$, $\mathfrak{D}$, and $\mathfrak{U}$. Convergence is typically rapid for the first few iterations, and then gets progressively slower; the asymptotic rate of convergence is linear.

The advantage of this procedure is that at each iteration only a triangular system of equations has to be solved. Moreover, the improvement of the speed of convergence can be achieved by a vestige of sparse matrix technology, i.e., the permutation of A into a form which is nearly triangular. The disadvantage of this procedure is its weak convergence. In some cases, if convergence is achieved, it is only linear. That is, if M has an eigenvalue of modulus near to 1, it may take many iterations to reduce the error by an order of magnitude. In a simulation context, this can lead to inaccurate solution of (3) and the concommitant forfeiture of the stability properties of (2). On the other hand, in many applications, A is diagonally dominant and symmetric on physical grounds. In this case, the eigenvalues of M have modulus strictly less than 1 and convergence is guaranteed.

B. Decomposition of Nonlinear Systems

Tearing decomposition at the nonlinear equation level is achieved in some third generation simulators by one of the following two approaches. The first approach solves the original system (1) by a Newton's method (e.g., equation (5) or a variant), and relies on the decomposition methods described earlier for linear systems. The SLATE program [11], being developed at the University of Illinois for IC simulation utilizes this approach, as does the chemical engineering simulation program of Westerberg and Berna [16]. The second approach decomposes the system at the nonlinear level by introducing additional iteration loops into the original Newton's method. The multilevel Newton method [17], used in IBM's MACRO IC simulator, which utilizes this approach, is discussed in Section IV later. A different example of the latter approach is the complementary pivoting algorithm [28], employed in the piecewise linear IC simulator reported by van Bokhoven [9].

Other third-generation simulators utilize temporal decomposition by solving the original system (1) with a nonlinear relaxation technique [19]. The MOTIS program [4], uses a nonlinear Gauss–Jacobi method, while the SPLICE program uses a nonlinear Gauss–Seidel–Newton iteration. These techniques are discussed in detail in Section V.

C. Dynamic System Decomposition

By dynamical system decomposition we mean the independent time-domain analysis of the subcircuits of a given circuit.

Most large IC exhibit temporal sparseness or latency [18], i.e., most of the subcircuits are inactive most of the time. Logic simulation [13]–[15] exploits this fact by using event scheduling and selective trace algorithms, which are possible due to the inherent delay unidirectional transmission characteristic of most of the logic gate models and to the absence of local feedback paths. Selective trace algorithms process logic gates only when they are active, i.e., when their internal variables or their inputs change. An impressive saving in simulation time can be obtained in cases where the restrictions imposed on the logic gate models prevent the numerical stability problems associated with "stiff" systems. In standard simulation selective trace algorithms cannot be applied since feedback paths are of importance and consequently the integration algorithms constrain all the variables of the circuit to use the same step size, necessarily the smallest one required to maintain truncation errors within limits.

Simulators such as SPLICE [5], MACRO [17], and SAMSON [7] achieve decoupling of the subnetworks and allow each subnetwork to follow its time trajectory at a self-tailored pace. These simulators assume that the subnetwork block structure is specified by the user in the input language. This permits the possibility of choosing different time steps for each subnetwork, allowing the sluggish subcircuits to take large steps independently of the rapidly changing subnetworks. Therefore, each subnetwork has to be scheduled for analysis at different time points and synchronization problems arise. MACRO and SAMSON process a subnetwork (i.e., solve the corresponding discretized equations) scheduled for analysis at a time point by extrapolating the variables associated with other subnetworks interacting with the subnetwork to be analyzed.

Scheduling algorithms are nontrivial: SPLICE, MACRO, and SAMSON use different strategies. In Section VI a prototype scheduling algorithm will be discussed together with a more detailed analysis of latency.

While the time decomposition discussed above is achieved by decoupling the subnetworks at the linear (SAMSON) or nonlinear (SPLICE and MACRO) equation level, a new iterative technique related to relaxation decomposition has been very recently introduced [7b] by Ruehli *et al.* In this method, the circuit equations are decoupled at the ordinary differential equations level before they are actually discretized. We shall refer to this method as the waveform relaxation decoupling (WRD) method. The WRD is better explained describing its application to a circuit described by its state equations:

$$\dot{x} = f(x, t), \qquad x(0) = x_0. \qquad (12c)$$

WRD Procedure:

For $v = 1, 2, \cdots$,
  Begin
  For $i = 1, 2, \cdots, n$
    Begin
    solve from $t = 0$ to $t = T$,
    $\dot{x}_i^{v+1} = f_i(x_1^{v+1}, \cdots, x_{i-1}^{v+1}, x_i^{v+1}, x_{i+1}^{v}, \cdots x_n^{v}), x_i^{v+1}(0) = x_i^0.$
    End
  $\delta^{v+1} \equiv \max_i \max_t |x^{v+1}(t) - x^v(t)|$
  End.

The iteration stops when δ^{v+1} is suitably small. Note that each component of the decomposition is processed for the entire time evolution individually and in a fixed sequence. The components which drive the component being processed, as well as the component which load this component, are handled by storing the temporal waveforms computed for those components on their most recent iteration. Then, when it is required to know how these components drive or load the component being processed, the necessary information can be obtained by interpolation on the stored waveforms of the adjacent components.

The preceding example shows the algorithm applied pointwise. Block WRD methods can also be derived in analogy with block relaxation methods for the solution of linear and nonlinear algebraic systems of equations. The blocks may be either be specified *a priori*, or determined algorithmically. The latter alternative is an open problem, perhaps amenable to approaches like those described in Section III.

In contrast to the conditions described above for the convergence of relaxation methods for linear and nonlinear systems of equations, the conditions for the convergence of the WRD method are quite mild. Remarkably, Lelarasmee [7c] has shown that convergence of the WRD method is guaranteed for any arbitrary piecewise continuous set of initial waveforms. Convergence will of course be linear like other relaxation methods. For circuits which cannot be described explicitly by state equations of the forms (12c), Lelarasmee [7c] has given conditions under which the procedure converges. Importantly, it is possible to show that for MOS circuits with a grounded capacitor at each node, convergence is guaranteed for any arbitrary piecewise continuous set of initial waveforms for the node voltages of the circuit.

III. Block Decomposition—Algorithms and Applications

In some of the literature on circuit simulation [5]-[7], [10], hierarchical decomposition is customarily identified directly by the "block diagram" implied by the nested models and subcircuits of the input description, as illustrated in Fig. 4. In this example some elements, e.g., resistors, diodes etc., are given explicitly in Fig. 4(a), while others, such as the large differential amplifier element, represent an entire circuit block, in this case that of Fig. 4(b). The transistor symbols of Fig. 4(b) are, in turn, symbols for the Ebers-Moll BJT model of Fig. 4(c).

If the block structure is not specified *a priori*, one must use an algorithm for determining an appropriate structure, such as the structure illustrated in Fig. 4(d). The appropriate structure is often defined in terms of the so-called dependency matrix of the system. The dependency matrix of a given (linear or nonlinear) system, denoted by A, is defined by

$$A_{ij} = \begin{cases} 0, & \text{if equation } i \text{ independent of variable } j \\ 1, & i, j = 1, 2, \cdots, n, \text{ otherwise.} \end{cases}$$

Note that the dependency matrix of a linear system represents the zero–nonzero pattern of the coefficient matrix and that the dependency matrix of a nonlinear system that of its Jacobian matrix. The important structures which appear in the literature to date are BBD, BBT, BLT, as defined in Section I. As defined earlier, the problem of obtaining the desired structure is that of finding permutation matrices P and Q such that PAQ has the appropriate block structure. By appropriate, we mean that the blocks are not too large, and the overall computational cost of solving the decomposed system is within specified limits. Two significant parameters of the decomposition are $n_{\max}$, the size of the largest block, and $q(PAQ)$, the size of the border. We observe, in general, that the computational complexity of solving the decomposed system is mainly related to $q(PAQ)$, and the ability to solve the decomposed system with limited storage and/or with parallel processing is strictly related to $n_{\max}$. Following [29b], the problem of obtaining the best decomposition may be stated in abstract terms as follows:

$$\text{P: } \min_{P,Q} \{q(PAQ) \mid PAQ \in A_{\text{BBD}}(A_{\text{BBT}}, A_{\text{BLT}})\} \tag{13}$$

subject to

$$n_i \leqslant n_{\max}, \quad i = 1, 2, \cdots, m(PAQ).$$

Here $m(PAQ)$ stands for the number of blocks in the decomposition.

We shall discuss the BBD, BBT, and BLT cases in turn. However, we first note that the preceding optimization problem is *NP* complete in each case. Thus our discussion of the cases will emphasize heuristic algorithms which are believed to produce near optimal solutions of (13).

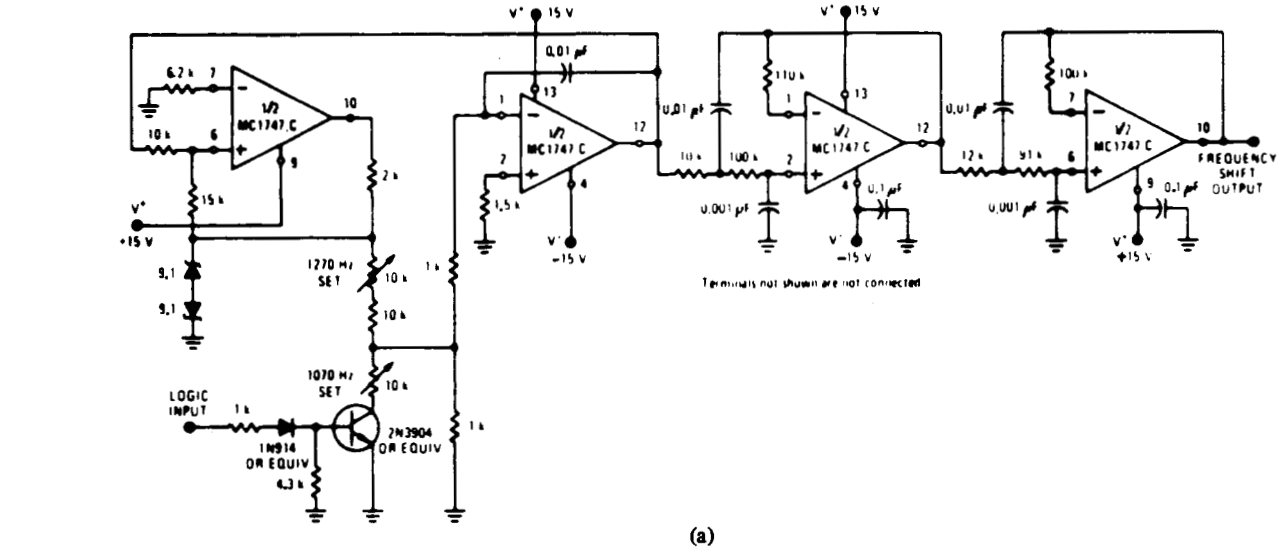
A. BBD Decomposition

We now discuss some algorithms for determining a BBD structure for a given dependency matrix. One such method [30] provides heuristics for determining blocks of nearly equal size, while minimizing the number of variables in the border. This method has been implemented in a program which has been used in power simulation [32], [33] and in parallelization of computer architectures. In this algorithm, $P \equiv Q^T$, where the superscript denotes transpose. The algorithm operates on the following principle. Assume that the dependency matrix A is structurally symmetric, and is represented by an undirected graph G, in which the nodes are in one-to-one correspondence with the variables (or equations) of the original system and the branches are in one-to-one correspondence with nonzero elements of the matrix. Let Z stand for a set of nodes which are candidates to form a block of the decomposition. Let S stand for a set of nodes of G which we call the *separator* of Z with respect to the graph G. Let W stand for the remainder of G, i.e., the part separated from Z by S. By definition, the sets Z, S, and W, are disjoint. The algorithm of [30] determines a sequence of nodes of G which are added to Z. For each node in this sequence, the cardinality of Z is incremented by 1. After each addition the separator S is updated. The algorithm monitors $|S|$ as the sequence is extended, thus obtaining the so-called "contour" of the sequence. Let Z_i denote the ith candidate block in the sequence, and assumed the sequence is of length k, where $|Z_k| \equiv n_{\max}$. Then Z_μ is selected as a block of the decomposition from the candidate blocks Z_i which satisfy

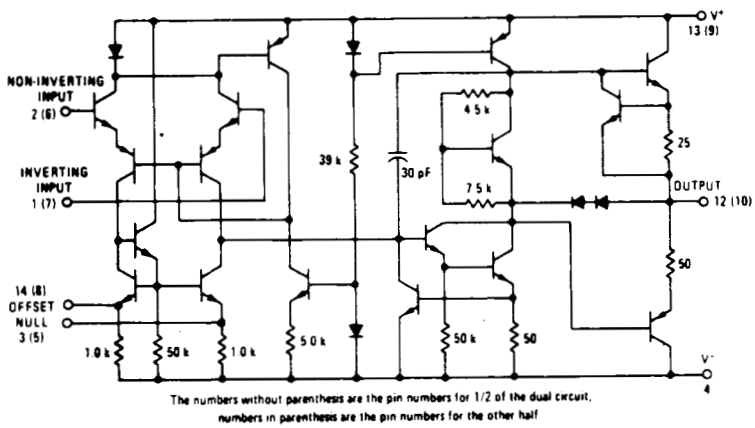
$$\mu \equiv \text{argmin } \{|S_i| \mid \alpha n_{\max} \leqslant |Z_i| \leqslant n_{\max}\} \tag{14}$$

where $|S_\mu|$, is the cardinality of the corresponding separator of the block Z_μ, and where α is a parameter ranging between 0 and 1.

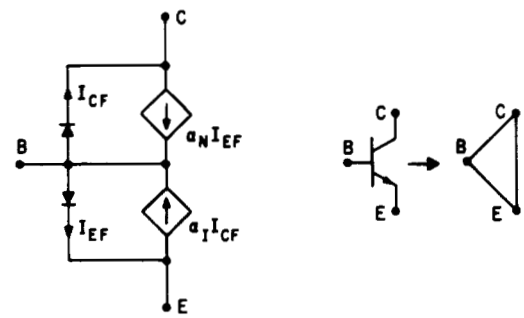
The above process is illustrated in Fig. 5. Fig. 5(a) shows the linear growth of $|Z_i|$ as a function of i. Fig. 5(b) shows the corresponding contour plot of $|S_i|$ for the case that the block is selected by the constraint in (13). The horizontal lines,



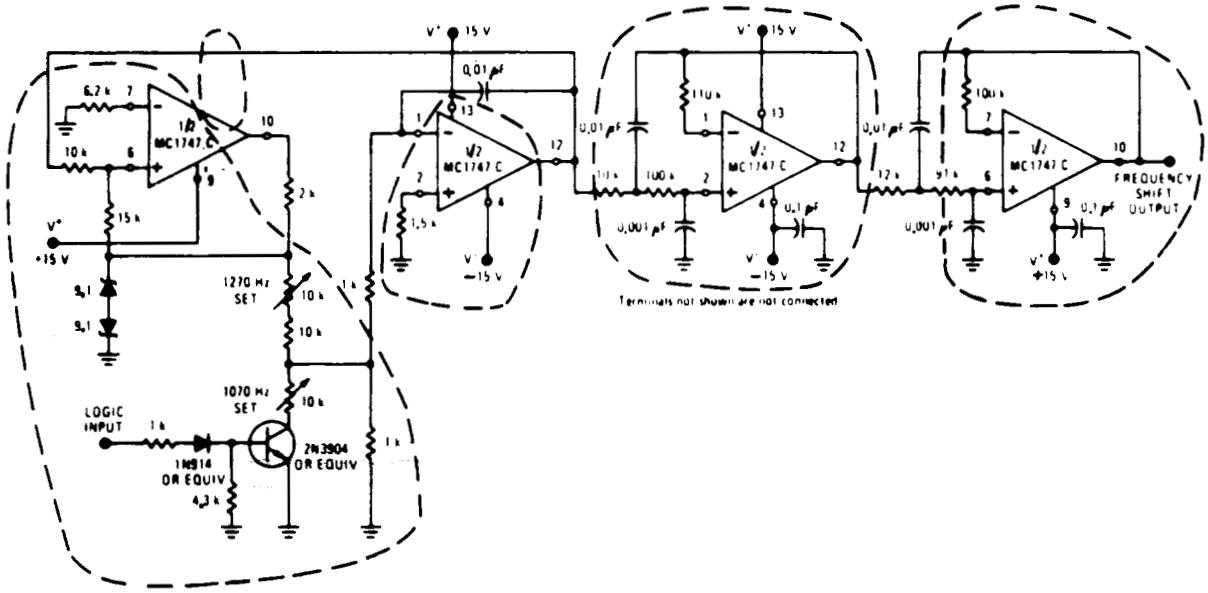
(a)



(b)



(c)



(d)

Fig. 4.

parameterized by α and $n_{\max}$, determine the set of i which satisfy (14). In Fig. 5(b), the separator of minimum cardinality S_μ occurs at the extreme right.

The algorithm may also select a "natural" block of the decomposition if it discovers a small separator between two large parts of the graph G. This process is illustrated in Fig. 5(c), which shows a contour plot which exhibits a strong minimum of $|S_i|$ at $i = \mu$.

The success of the algorithm depends on the heuristics chosen for initializing and extending the sequence. The

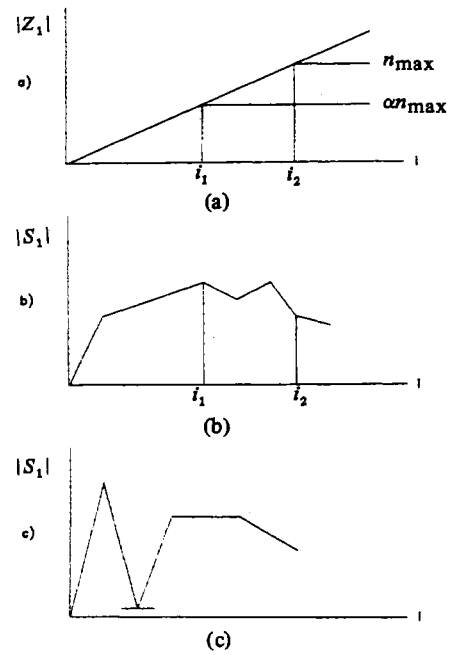


Fig. 5. Contours of block size and separator size.

heuristic for extending the sequence is to select the node for which the increase in $|S|$ incurred by the selection is minimized. Note that the "increase" is usually negative as the contour begins to identify a "natural" cluster. Thus this algorithm can be categorized as a "greedy algorithm," in the sense of Edmonds and Lawler [33].

The preceding algorithm has been applied in the example of Fig. 6. The rows of the matrix of Fig. 6(b) represent the equations describing the 30-bus electrical power distribution system shown in Fig. 6(a). The columns represent the (complex) power variables of the system. The row and column labels represent the permutations required to place the matrix into BBD form. The variables 4, 20, 24, and 27 are identified as the border variables of the decomposition. The symbol "x" represents the dependency of row (equation) on column (variable). The heavier lines represent the partitioning of the diagonal block variables.

Another powerful technique, which is related to the above algorithm, is called "nested dissection," and is due to George [25], [26]. This technique can be used for decomposition purposes as well as for the conventional purpose of ordering the rows and columns of a sparse matrix in order to achieve an economical LU factorization. In the decomposition context, the algorithm may be thought of as follows. Let Z, S, and W be defined as above. Again consider a sequence Z_i, $i = 1, 2, \cdots$. In the case of nested dissection, the initial objective is to determine i such that

$$|Z_i| \cong |W_i|. \tag{15}$$

Once this objective has been attained, the algorithm is recursively applied to the subgraphs corresponding to Z and W instead of the original graph G. A significant difference between this algorithm and the BBD algorithm described earlier is that here the sequence is extended not with a single node but with the entire set $H \subset W$, where H is a subset of the set of nodes ADJ(S), which we shall henceforth define as the set of all nodes adjacent to one or more nodes in S. If the recursive "outer loop" of this algorithm is applied k times, the algorithm will have dissected the original graph into 2^k blocks, i.e., subgraphs. However, in contrast to the previous algorithm, the blocks produced by nested dissection decrease in size from top to bottom of the BBD. This is due to the removal of the separator sets from the blocks.

The remarkable thing about the nested dissection technique is that when used as an ordering algorithm, it produces proven optimal multiplication counts for point Gaussian elimination for dependency matrices corresponding to or related to the Laplacian operator in two or three dimensions. It is widely believed that nested dissection produces near-optimal results for any system whose dependency matrix is characterized by a two-dimensional graph, e.g., a two-dimensional array (not necessarily regular) of logic circuits.

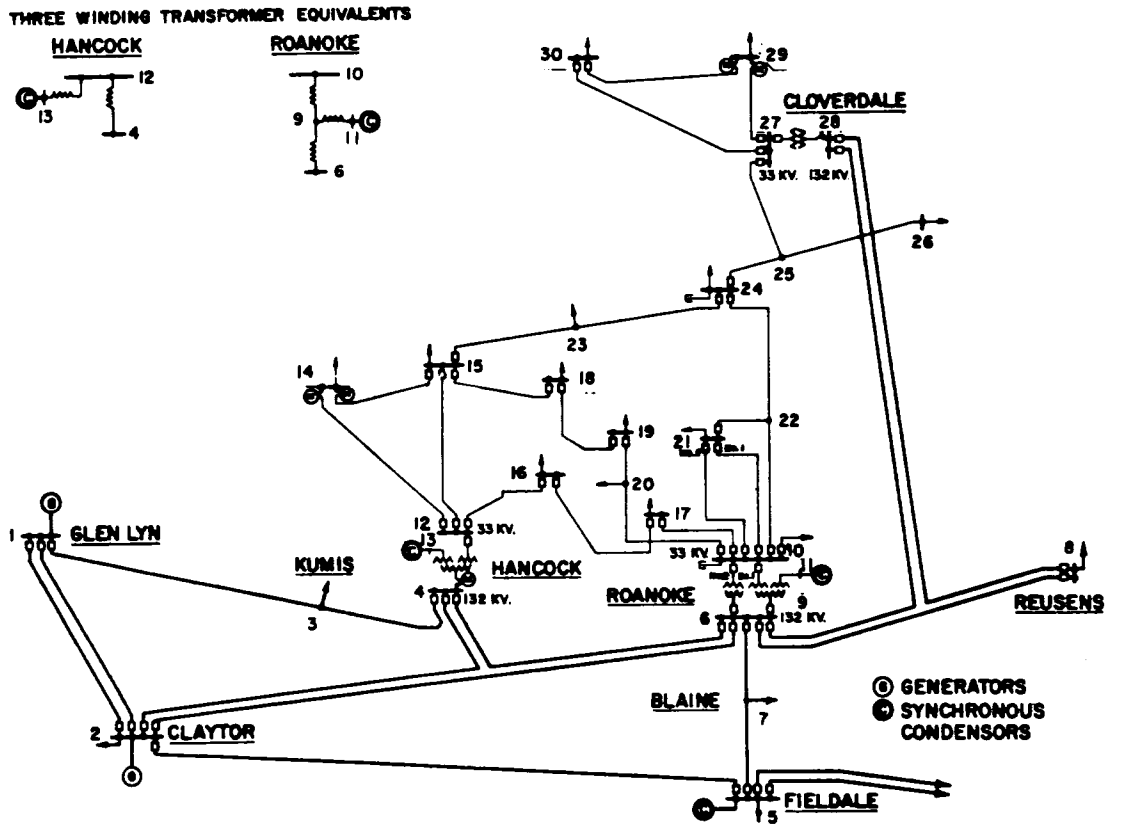
The nested dissection process is illustrated in Fig. 7. In Fig. 7(a), the graph G of the dependency matrix of the finite difference equations of the Laplacian operator discretized on a 9×9 grid is given. The process begins by heuristically selecting a "seed," which for our example is chosen to be the node in the middle of the left boundary of the graph. We initialize Z_1 to consist of this seed, and set $S_1 = \text{ADJ}(Z_1)$. The sequence is extended as described above. The cardinality of Z_i is noted at the left of Fig. 7(a). A block of the decomposition is selected when condition (15) is satisfied. The nested dissection process then continues recursively, dividing each currently largest block in the decomposition into two approximately equal parts. The success of the algorithm depends on the shape of the grid graph and on the heuristic selection of the seed node. For the example of Fig. 7(a), this process leads to the graph decomposition of Fig. 7(b), where the symbols S_μ, $\mu = 1, 2, \cdots$ label the separators chosen at each stage of the recursion.

This technique has had spectacular success in the area of semiconductor device simulation [34], [35]. Most sparse matrices which arise in this application area have nonzero patterns which are variants of the well-known 5-diagonal pattern of the Laplacian operator in two dimensions. Fig. 8(a) illustrated the Laplacian matrix of 16 unknowns, corresponding to a 4×4 grid graph. Fig. 8(b) shows the result of applying nested dissection. Note that the nested-dissection strategy leads to diagonal blocks of the BBD form which decrease in size from top to bottom, in contrast to the uniform block sizes produced by the first algorithm discussed in this section. Still, the nested-dissection technique has been proven to give a multiplication count for this nonzero pattern [36], which is asymptotically optimal for arbitrarily large grid graphs. George [26d] has also published a "one-way" version of nested dissection which does tend to produce uniform block sizes.

The application of the technique to VLSI-size networks remains a promising avenue for future exploration.

B. BBT Decomposition

The alternative decompositions addressed by (13) yield the BBT and BLT forms. Actually, either of the algorithms described above for the BBD case could be appropriately modified to operate on the bipartite graph in the unsymmetric case to produce such a form. The key modification would be to identify the separator set exclusively with the "right" (i.e., column) nodes of the bipartite graph. The selection and exploitation of the resulting block structure appears to be an-



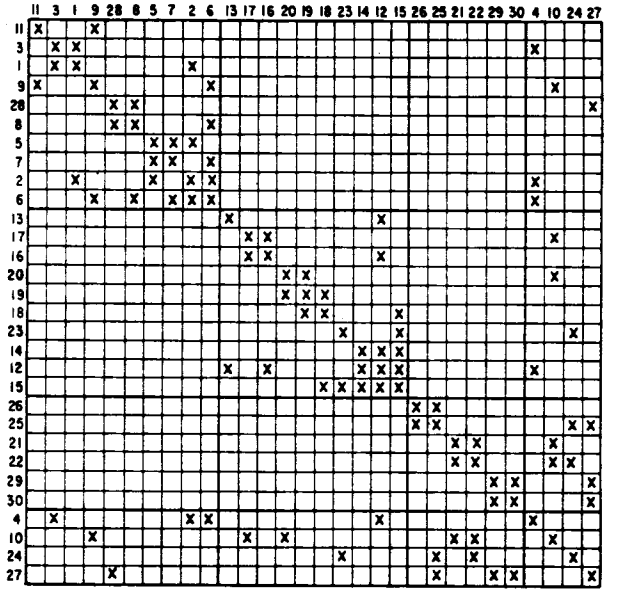


Fig. 6. AEP bus test system bus code diagram.

other promising avenue for future research.

There is, however, a substantial literature pertaining to the selection of a BLT structure, which is a special case of the BBT for which $n_{\max} = 1$. This case has important applications and has been extensively studied. To start with we define an assignment as a set of distinct row and column index pairs (r_i, c_i), such that $A_{r_i,c_i} \neq 0$, $i = 1, 2, \cdots, n$. That is, the assignment permutes A into a matrix with a nonzero diagonal. We can divide algorithms for obtaining a BLT into two classes: 1) those operating on a fixed assignment (i.e., $P \equiv Q^T$, which means that the assignment itself is not exploited as a degree of freedom in minimizing the size of the border), and 2) those which exploit the assignment for this purpose.

In the first category are algorithms by Roth [37], Hellerman and Rarick [38], Kevorkian [39], [40], Cheung and Kuh [41], Guardabassi [42], and Smith and Walford [43]. In this case

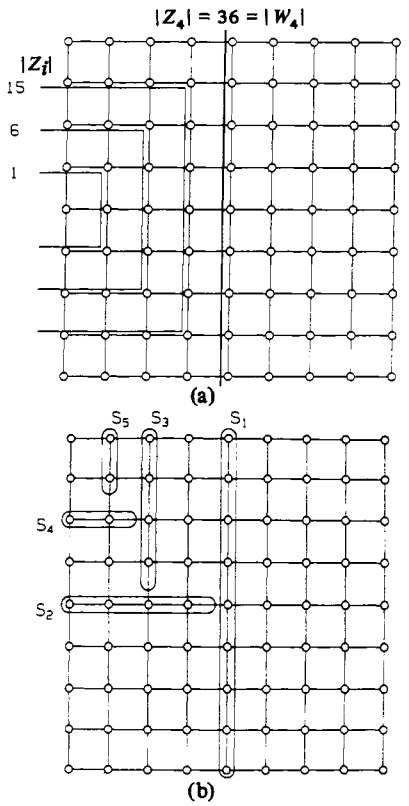


Fig. 7. (a) Selection of first separator. (b) Separator sequence (first 5 only).

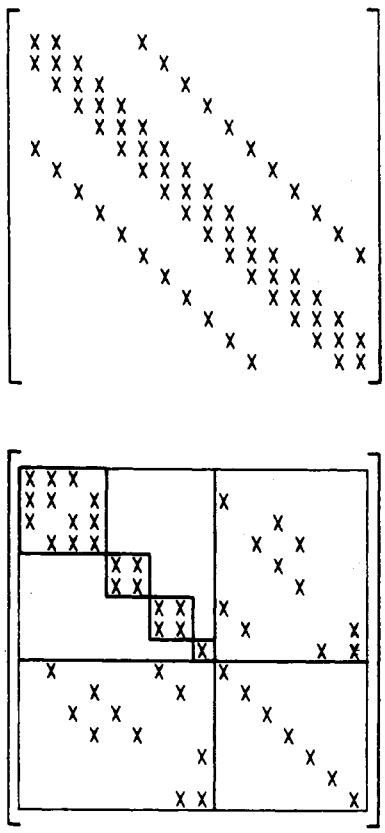


Fig. 8. BBD structure from nested dissection.

the optimization problem (13) is equivalent to the problem of minimizing the cardinality of the essential (feedback) vertex set for directed graphs. The algorithms of Roth, Cheung and Kuh, and Smith and Walford were motivated by logic simulation and logic "design for testability" [37], applications in which it was desired to find a way in which the minimal set of registers could convert a sequential logic circuit into an equivalent combinational form. The algorithm of Kevorkian was used to simulate the design of a chemical plant. The algorithm of Hellerman and Rarick was used for updating the basis for a linear programming package. This latter algorithm is, therefore, important in piecewise linear circuit simulators such as those reported by Katzenelsen [44], Kuh [45], and Von Bokhoven [9]. Due to the importance of these applications, we offer a brief discussion of some of the algorithmic techniques used to obtain an effective bordered triangular form.

The problem addressed by algorithms in this category is *NP*-complete. Therefore, algorithms which seek to find the exact optimum will have exponentially bounded computational complexity. The Hellerman–Rarick algorithm, proposed for linear programming applications, proceeds heuristically to find a nearly optimal BLT form.

The other algorithms in this category deal with the directed graph representation of the dependency matrix. All proceed by reducing the size of the given graph by transformations which do not affect the cardinality of the minimum essential set. After the graphs have been thus reduced to a minimal size, the minimum essential set itself is determined by a branch and bound algorithm. Thus the exact minimum is obtained, rather than an approximation. However, all these algorithms are, consequently, exponentially bounded.

Algorithms in the second category, which attempt to optimize PAQ by heuristically choosing the assignment, have been presented by Hellerman and Rarick [46], Sangiovanni and Bickart [47], and Jess and Trouborst [8]. These algorithms have a strong potential for use in circuit simulators implemented for on relatively small machines such as minicomputers and desktops.

The algorithm of Bickart and Sangiovanni operates on the bipartite graph of the dependency matrix, and so is conceptually based on graph theory. However, we discuss this algorithm in terms of both the dependency matrix and the corresponding bipartite graph. The algorithm proceeds by selecting a maximal sequence of nonzero pivots from the irreducible blocks of the dependency matrix. The sequence is extended by two mechanisms. The main mechanism is to enumerate the rows which have minimum count in the "unpivoted" portion of the matrix. Among these rows, the column of maximum count in the unpivoted portion is selected, along with a corresponding row in the "minimal row count set." This heuristic selects the minimum row count because a nearly lower triangular form is sought. The maximal column count is motivated by a desire to process the hardest columns, i.e., the columns which provide the greatest opportunity for creating cycles (which would increase $q(PAQ)$). The second mechanism identifies submatrices for which an optimal border may be found by a process of elimination of cycles of length two in the corresponding bipartite graph.

Note that this algorithm creates a BLT in which the diagonal is populated solely by elements which were nonzero in the original matrix. This strategy promotes numerical stability, as

documented by Brayton *et al.* [48], who showed that pivoting on originally zero valued elements can lead to exact symbolic cancellation.

Jess and Trouborst [8] have presented an algorithm which is both a variant and an extension of the Bickart–Sangiovanni algorithm. This algorithm introduces another degree of freedom to be exploited in minimizing $q(PAQ)$. In their algorithm, equations in the original system (and, therefore, rows of the dependency matrix) which correspond to linear equations are identified. Then, equations of this type are subjected to Gaussian elimination on originally nonzero pivots to further reduce $q(PAQ)$. This approach to simulation also resembles that of Moad [49], who attempted to find an "evaluation order" for the equations and unknowns.

IV. Exploiting a Given Topological Decomposition

In this section, we discuss various ways to exploit a given topological decomposition (such as that provided by the methods described in the previous section). Section IV-A treats the necessary step of selecting the variables which interact between the various hierarchical levels of the decomposition. Section IV-B is devoted to a treatment of a numerical method for exploiting decomposition at the nonlinear equation level to computational advantage while retaining the stability and convergence properties of standard simulation methods.

A. Selecting the Exogenous Variables of a Decomposition

Even if an *a priori* "functional" hierarchy is chosen for the decomposition, one must still decide which variables in the functional block are "endogenous," i.e., interact only inside the block, and which are "exogenous," i.e., interact with the variables at the next level of the hierarchy. In electrical networks, for example, one must decide whether the branch currents or voltages are chosen to be the exogenous variables, or the external node voltages of the functional blocks are chosen to be exogenous.

In the block diagram approach, the exogenous variables are not determined algorithmically but are selected by the designer of the simulation program. This choice characterizes the type of decomposition actually employed. For example, if the exogenous variables are chosen to be the voltage on the nodes external to the differential amplifier model in Fig. 4(a), the well known case of "node tearing" results [5], [7], [10], [11]. On the other hand, if the exogenous variables are chosen to be the branch currents, the case of branch tearing is selected [27]. Note that with either of these approaches, the selection of the appropriate formulation of the network equations is made after the tearing is performed, and remains as a degree of freedom to be exploited in obtaining the most economical simulation. Hajj [27] has studied both branch and node tearing and pointed out an equivalence between this type of tearing and the block factorization approach for solving linear algebraic equations (cf. the discussion of (8) in Section II-A). In addition, he enumerated the various types of block factorization which are possible and pointed out that different combinations of block factorization techniques are appropriate for different blocks. Also, he showed that for a specific case, using the best technique for each block, node tearing was slightly superior to branch tearing. Sangiovanni *et al.* [29], [30], gave a more general result comparing these two methods. They proved under quite mild topological restrictions, that node tearing always gives a smaller number of exogenous variables in the decomposed system. The node tearing approach has been used in the SPLICE program [5], and in the SLATE program [11]. For example, the SPLICE program input is a mixture of subcircuits specified as models and as conventional circuit elements. Since SPLICE is a mixed mode simulator, some of the models are analyzed with simplified "timing analysis" techniques [4], and some are analyzed with full-fledged circuit analysis. Models which are specified for circuit analysis are identified by special delimiters. The contact nodes of these models are chosen as the node tearing variables, and thus become the exogenous variables of the decomposition. Similar techniques are used in the SLATE program [11].

B. Exploiting a Given Decomposition at the Nonlinear Level

An effective algorithm for exploiting a BBD structure explicitly specified by the input language of a circuit simulator such as SPICE, SPLICE, or ASTAP, is the multilevel Newton–Raphson algorithm of Rabbat, Sangiovanni, and Hsieh [17]. In general, each of the subcircuits specified in the input language interacts with the rest of the circuit only through a small number of exogenous variables $u \in R^k$. The endogenous variables of one of these subcircuits can be partitioned into two sets. The first set, called the output variables, is denoted by $y \in R^k$, and are in one-to-one correspondence with the exogenous variables. For example if the exogenous variables are chosen to be the node voltages, then the set y corresponds to the currents entering the subcircuits. The second set, called the internal variables, will be denoted by the vector $x \in R^m$.

Assume that the circuit to be simulated is static, i.e., is described by a set of nonlinear algebraic equations. In general, given the values of the exogenous variables, the output variables and the internal variables are determined by a system of equations of the form

$$H(u, x, y) = 0. \tag{16}$$

Given u, the interaction of the subcircuit with the rest of the circuit is completely described by y. Thus to simulate the "super" circuit, it suffices to compute y for each subcircuit.

Assuming that (16) has one and only one solution for each u, (16) describes implicitly a map from u to y. This map is denoted by $G_y(u)$ and is called an *exact macromodel*. We shall now discuss an algorithm for analyzing a network with subnetworks described by equations of the form (16), [17].

For the sake of simplicity, we assume that only one subcircuit is described by its macromodel. Let the equations of the network be written as

$$F(u, G_y(u), w) = 0 \tag{17}$$

where $w \in R^p$ is the vector of network variables in the network not interacting with the subcircuit, and G_y represents the macromodel of the subcircuit. Newton–Raphson's algorithm applied to (17) consists of the following scheme:

$$D_u F(u, G_y(u), w)\,\Delta u + D_G F(u, G_y(u), w)\,DG_y(u)\,\Delta u + D_w F(u, G_y(u), w)\,\Delta w + F(u, G_y(u), w) = 0. \tag{18}$$

Here D_z represents the derivative with respect to the generic variable z.

Thus, to apply Newton's method, we need to evaluate $G_y(u)$ and $DG_y(u)$. Recall that the macromodel $G_y(u)$ is

implicitly determined by the nonlinear system of equations

$$H(u, x, y) = 0. \tag{19}$$

To evaluate $G_y(u)$, we can use a second Newton process on (19) which yields

$$D_{x,y}H(u, x, y)(\Delta x, \Delta y)^T + H(u, x, y) = 0. \tag{20}$$

This second Newton process is at a lower level since u is determined from (18) and held fixed in (20). Now, if (19) is solved precisely, then the error in the evaluation of the macromodel and its derivative is zero and when these are used in (18), we have a true Newtonian iteration with local quadratic convergence. However, if the macromodel and its derivative are not determined precisely, then the question of quadratic convergence is open. The idea proposed in [17] is to retain local quadratic convergence in the presence of error as follows.

It would seem to make no sense to solve (19) to a higher precision than the current iteration for (17), and it would seem only necessary to tighten the convergence control for (20) at the same rate (18) is converging. In the algorithm proposed in [17] iteration (20) is stopped whenever

$$\|\Delta x, \Delta y\| \leqslant \|\Delta u, \Delta w\|^2. \tag{21}$$

In [17], it is proven that this algorithm has local quadratic convergence. Note that if iteration (20) is stopped according to a different criterion, for example, whenever $\|\Delta x, \Delta y\| \leqslant \|\Delta u, \Delta w\|$, the algorithm can be shown to converge but loses its quadratic convergence property.

Like other decomposition methods, this algorithm permits individual subcircuits to be processed in parallel. In this case quadratic convergence is retained. The method also has two other principal advantages. First, if there are identical subcircuits, then the linear equation solution step (20) for each identical subcircuit may be obtained with the same (symbolic) LU factors. Second, note that although we have discussed the algorithm of [17] in the context of a two level hierarchy, the method applies equally well to a multilevel hierarchy, which is the most typical case in IC simulation.

V. Relaxation Decomposition of Nonlinear Systems

In this section, we will discuss a second avenue by which the decomposition of nonlinear systems has been approached in the literature of the engineering disciplines, especially the circuits literature. While the first approach, presented in the Sections II and IV, derives conceptually from the "direct method" approach to solving linear algebraic systems, the second derives conceptually from the classical subject of relaxation methods.

The second approach is characterized by the inclusion of more or less classical relaxation methods. In the integrated circuit context, this approach began with the work which led to the MOTIS simulator, [4]. MOTIS was a revolutionary simulator in three main respects:

1) It limited severely the types of networks it dealt with (MOS devices with quasi-unidirectional circuit models, and a grounded capacitance on every node);
2) it discarded both sparse Gauss elimination and conventional Newton–Raphson as a solution method;
3) it discarded implicit time integration methods such as backward Euler, trapezoidal rule, or stiffly stable methods.

One of the key contributions made by the authors of MOTIS was the physical reasoning devoted to justifying the method used for advancing the time step, and, in fact, selecting the time step taken. The physical reasoning was indeed well founded, and the success of MOTIS, as implemented, was a landmark for the CAD area.

However, as subsequent investigators soon found out, MOTIS, as implemented, was not optimally efficient and, in fact, had problems.

In order to discuss these problems we express the nodal equations of a general network as follows.

$$J(v) + C\dot{v} = 0. \tag{22}$$

Here v stands for the vector of node voltages, C is the (assumedly linear) capacitance matrix, and $J(v)$ is the vector of currents feeding the capacitors. Throughout this section, we will assume that there is a capacitor to ground from every node. For purpose of exposition, we will also discuss only the case where the backward Euler method,

$$\dot{v}_{n+1} \equiv (v_{n+1} - v_n)/h \tag{23}$$

is used to discretize the time derivative operator. Here the subscript denotes the time point of the integration and $h \equiv t_{n+1} - t_n$. For simplicity, we shall henceforth drop the subscripts referring to the time point.

Because we shall be describing the literature in this area in terms of *point* relaxation methods [19], we must consider solution methods which sweep through the component equations of (22), solving one equation at a time for one unknown at a time. A procedure for such a sweep was given in (12). However, in order to compare the sweeps used in MOTIS, MOTIS-C, and SPLICE, we need a different notation. The problem is that we need to distinguish between three distinct classes of components of the node voltage vector, namely, $\hat{v}_j$, the variable being solved for in the jth equation, $\tilde{v}_i$, $i < j$, a variable already solved for during this sweep through the equations, and the variables remaining to be solved for, $\bar{v}_k$, $k > j$, $j = 1, 2, \cdots, q$, $q \equiv |v|$. Given this complication, we shall find it useful for our discussion of relaxation methods to introduce the functions

$$j(\tilde{v}, \hat{v}, \bar{v}) \equiv \begin{matrix} j_1(\hat{v}_1, \bar{v}_2, \cdot \bar{v}_q) \\ j_2(\tilde{v}_1, \hat{v}_2, \cdot \bar{v}_q) \\ \cdot \\ j_q(\tilde{v}_1, \tilde{v}_2, \cdot \hat{v}_q) \end{matrix} \tag{24a}$$

and

$$\begin{aligned} &c(\tilde{v}, \hat{v}, \bar{v}, v_n, h) \\ &\quad \equiv (1/h) \begin{matrix} C_{11}(\hat{v}_1 - v_{n,1}) + C_{12}(\bar{v}_2 - v_{n,2}) \cdot + C_{1q}(\bar{v}_q - v_{n,q}) \\ C_{21}(\tilde{v}_1 - v_{n,1}) + C_{22}(\hat{v}_2 - v_{n,2}) \cdot + C_{2q}(\bar{v}_q - v_{n,q}) \\ \cdot \\ C_{q1}(\tilde{v}_1 - v_{n,1}) + C_{q2}(\tilde{v}_2 - v_{n,2}) \cdot + C_{qq}(\hat{v}_q - v_{n,q}). \end{matrix} \end{aligned} \tag{24b}$$

Note the triangular structure of these functions, whose relationship to the original functions $J(v)$ and $C\dot{v}$ is defined so that

$$j(v, v, v) + c(v, v, v, v_n, h) = 0, \tag{25}$$

with the substitution (23) for the backward Euler formula, is equivalent to (22).

In MOTIS [4], these equations were solved approximately

with a one-step "time advancement" scheme which consists of a single relaxation sweep through the "diagonal" equations

$$j_i(\bar{v}, \hat{v}, \bar{v}) + c_i(\bar{v}, \hat{v}, \bar{v}, v_n, h) = 0, \quad i = 1, 2, \cdots, q. \quad (26)$$

Note that $\tilde{v}$ does not appear here, so only the diagonal terms in (24) are updated in any given sweep.

However, the nonlinear equations encountered at each step of a sweep through (26) are not solved exactly. Instead, a single *regula falsi*, [19], step is taken to approximately solve each equation. This linearizing approximation was motivated in MOTIS by clever physical reasoning based on first principles of MOS device physics. It is justified when sufficiently small time steps are taken. With hindsight it can be seen that the MOTIS algorithm can be regarded as a nonlinear version of the classical Jacobi "iteration" [19]. However, the term "iteration" is used advisedly, since only a single pass through the system (26) is taken. Since the MOTIS system does not actually solve (22), the original astable backward Euler implicit integration method can be shown to be reduced to an explicit form which is not astable. Thus the method has the disadvantage that the time step may be controlled not by user determined accuracy requirements, but by stability requirements, which may require a much smaller time step than that needed for accuracy. The avoidance of this situation was one of the prime motivations for the development of sparse matrix technology.

However, the MOTIS approach has advantages, some of which are:

1) that the computational expense of taking a single time step is very small;
2) that the equations are decoupled and can be solved in any order, perhaps on parallel processors;
3) that since after the *regula falsi* step, each equation is linear and a function of only a single variable, no sparse matrix software for Gaussian elimination is required.

In cases where the inputs to the circuit do not change very much over a sequence of time steps, the MOTIS time advancement method (26) can be viewed as equivalent to a Jacobi iteration. In this case the accuracy and stability of the time advancement scheme can be related to the convergence of the analogous Jacobi iteration. This iteration will converge well if the Jacobian matrix of the system (26) is sufficiently diagonally dominant [19a], [19c]. If we assume that the MOS circuit has no floating capacitors, the capacitance matrix C of (25) is a diagonal matrix. In this case the sufficient diagonal dominance may always be obtained for sufficiently small h. However, if the node capacitances are too small, an unacceptably small value of h may be required.

Moreover MOTIS has a problem with floating capacitors. If the floating capacitance is large with respect to the node capacitance, at any given node, there may be no value of h which makes the Jacobian sufficiently diagonally dominant to ensure convergence of the analogous Jacobi iteration. Thus the corresponding accuracy and stability of the MOTIS time advancement scheme will be insufficient for any value of h.

Other investigators [5], [14], [51], have expanded on the MOTIS concept by taking further advantage of the nearly unidirectional nature of MOS devices. In SPLICE [5], [14], the Jacobi-like formulation of (23) is replaced by

$$\bar{v} \leftarrow v_n \quad (27a)$$

$$j_i(\hat{v}, \hat{v}, \bar{v}) + c_i(\hat{v}, \hat{v}, \bar{v}, v_n, h) = 0, \quad i = 1, 2, \cdots, q \quad (27b)$$

$$\bar{v} \leftarrow \hat{v}. \quad (27c)$$

Note that in contrast to (26), $\hat{v}$ is substituted for the "lower triangular" variable $\tilde{v}$, instead of $\bar{v}$. Thus whereas (26) constituted q independent equations in 1 unknown apiece, equation (27) constitutes a triangular system of q equations in q unknowns, analagous to a back substitution process. This is the essential part of the well known Gauss–Seidel process [19]. In a true Gauss–Seidel process (27b) and (27c) would be iterated to convergence. Like MOTIS, SPLICE linearizes the nonlinear equation (27b), and takes only a single relaxation sweep through (27).

If the Jacobian of (27) with respect to the third argument v is identically zero, then it is in a degenerate form of Fig. 1(b) in which all block sizes are unity. As noted in Section II, equation (27) converges in a single sweep in this case. Note that this would be the case for the simplest MOS device models when there are no pass transistors. If this were indeed the case and if (27) represented a truly linear system, then the Gauss–Seidel iteration would converge in exactly one step. The corresponding time advancement scheme would then be astable, and the method would have no time-step limitation due to stability, regardless of how small the capacitances were. Note that this would not be true of the Jacobi iteration approach used in MOTIS, an indication of the greater power of the one-step Gauss–Seidel time advancement scheme.

Thus if there is no feedback in a given circuit, it can be concluded that the SPLICE formulation gives more accurate results (exact in the linear case) than the MOTIS formulation. In fact, with this approach, it is no longer necessary to have a capacitor to ground at every node. Note, however, that to take advantage of this desirable property, it is necessary to *order* the equations of (27), whereas with (26), the equations could be processed in an arbitrary order. Thus the Gauss–Seidel approach requires a vestigial form of sparse matrix preprocessing.

The MOTIS-C program, represents an intermediate approach where a Gauss–Seidel technique is employed, but the equations are not ordered into the "most lower-triangular" form. In this case, even if all the devices are unidirectional the Gauss–Seidel iteration will not converge in one step, so that some of its intrinsic power is lost.

The introduction of the Gauss–Seidel iteration into the MOTIS concept was a significant contribution toward dealing with the problem of small capacitances in MOS circuits. However, this improvement did not deal with the problem of floating capacitors. The presence of floating capacitors degrades the accuracy and stability of the basic MOTIS approach and furthermore introduces nonphysical oscillations into the solutions of digital MOS networks. When viewed as integration methods, the Jacobi and Gauss–Seidel iteration schemes described above can be said to introduce complex "parasitic roots" (i.e., eigenvalues of the difference equations which do not correspond to eigenvalues of the approximated differential equations).

One technique for dealing with these difficulties without giving up the essential character of the MOTIS–SPLICE approach was given in [52], [53]. The problem was identified as the destruction of symmetry (and therefore of the positive definiteness of the Jacobian) by the triangular aspect of the Gauss–Seidel approach. The basic idea was to replace the

"one-step" Gauss-Seidel solution method with a method that takes two half-steps instead. The method symmetrizes the basic Gauss–Seidel technique by taking one half-step in the normal "forward" (i.e., lower triangular) direction and a second half-step in the "backward" direction. The algorithmic aspects of their approach can be summarized as follows. Let $\hat{v}_i^{1/2}(\bar{v}^{1/2})$ stand for the variable being updated (not yet updated) in the ith equation of the first half-step. As above, let $\hat{v}(\bar{v})$ stand for components being updated (already updated) on the second half-step. Then the two-step Gauss–Seidel procedure can be written

$$\bar{v}^{1/2} = v_n \tag{28a}$$

$$j_i(\hat{v}^{1/2}, \hat{v}^{1/2}, \bar{v}^{1/2}) + c_i(\hat{v}^{1/2}, \hat{v}^{1/2}, \bar{v}^{1/2}, \bar{v}^{1/2}, (h/2)) = 0, \quad i = 1, 2, \cdots, q \tag{28b}$$

$$\bar{v} \leftarrow \hat{v}^{1/2} \tag{28c}$$

$$j_i(\bar{v}, \hat{v}, \hat{v}) + c_i(\bar{v}, \hat{v}, \hat{v}, \bar{v}, (h/2)) = 0, \quad i = q, q-1, \cdots, 1. \tag{28d}$$

$$v \leftarrow \hat{v}. \tag{28e}$$

In interpreting (28) it is important to observe a few key points. First note that except for the superscripts, (28a) and (28b) are identical to (27a) and (27b) for one-half of the time step h. Second, note that in (28d), the Gauss-Seidel step is reversed in two important respects:

1) the positions of $\hat{v}$ and $\bar{v}$ in the argument lists of the two functions are reversed,
2) the order in which the equations are processed in (28d) is reversed.

In concert, these two reversals amount to processing the physical devices in reverse order.

This method shows strong similiarity to the well-known alternating-direction implicit method [19b], as well as to a method proposed by Kahan [54]. Its advantages may be summarized as follows. First, integration with this method has shown good stability properties on practical examples. Second, this method is astable for a simple circuit consisting of a pi-section of resistors and capacitors, an advantage which cannot be claimed by the methods in MOTIS and SPLICE. Third, the method does not introduce complex parasitic roots, and therefore does not produce nonphysical oscillations and overshoot conditions.

A third variation on this basic approach is implemented in the program DIANA, [55]. DIANA considers the same types of MOS circuits as MOTIS and SPLICE. DIANA also employs the *regula falsi* linearization technique developed for MOTIS. Without stretching the truth too much, we can regard DIANA's time advancement technique as "block" Gauss-Seidel, where the block structure is obtained as follows. For the sake of simplicity we assume that like SPLICE and MOTIS, the DIANA circuit equations can be expressed in the form

$$j_D(v, v, v) + c_D(v, v, v, v_n, h) = 0 \tag{29}$$

where the subscript is added to note that DIANA uses a slightly different MNA formulation [3], to accomodate ideal switches. Thus, the DIANA algorithms are somewhat more complicated than discussed below. The algorithms in DIANA are similar to those described in [56]. The dependency matrix discussed in Sections II–IV is obtained from the sparsity pattern of the Jacobian of (29) with respect to v. Then the algorithm of Tarjan [57], [58] is used to find the block triangular (i.e., BBT form with zero border) structure of (29). This structure, like that of SPLICE, is triangular except for occasional square blocks, which correspond to what the authors of DIANA refer to as the blocks enclosed in "tightly coupled feedback paths." Physically, these blocks derive either from floating capacitors, pass transistors or explicitly intended feedback.

If we consider these blocks as units in a "block" Gauss-Seidel, then the resulting structure is triangular in the block sense of [56]. If the equations were linear, the block Gauss–Seidel method would converge in a single relaxation sweep. What is different from SPLICE and MOTIS is that the nontrivial blocks are handled explicitly by LU factorization. Thus DIANA has an even stronger tie than SPLICE to the sparse matrix technology employed in standard second-generation circuit simulators.

VI. Event Scheduling and Latency in Mixed Mode Simulation

In the preceding sections, we have discussed some of the numerical techniques that are used by mixed mode simulators such as SPLICE, SLATE, and DIANA. Emphasis was placed on decomposition methods as a means of paring a large computation down to reasonable size. In this section we describe some related techniques which are function specific to the task of "mixed mode" simulation. A mixed mode simulator is defined here to be one which is capable of simulating different parts of a given circuit or system in two or more modes in the following list:

Register Transfer Level Simulation
Logic Simulation
Critical Path Timing Simulation (PERT)
MOS Timing Simulation
Circuit Simulation.

As we enter the VLSI era, mixed mode simulation is becoming increasingly important. It is possible to meaningfully extend this list in both directions, but we feel this list is adequate in the context established for the present paper.

There are many ways to view the preceding simulation hierarchy, e.g., bottom-up, top-down, architecture-device, etc. However, for our purposes, a useful and equally valid way is to think of it as a feedback hierarchy. At the top levels, feedback is either nonexistent or plays a minor role. At the bottom level of conventional circuit simulation, feedback plays a very significant role.

The analysis performed by mixed mode simulators reflects this hierarchy. For example, SPLICE requires the user to specify which blocks of the circuit has to be analyzed at the logic, timing or circuit levels. Then, the blocks of the circuit declared by the user as logic blocks are analyzed with logic simulation algorithms which essentially ignore feedback; the blocks declared by the user as timing blocks are analyzed with the algorithm described in Section V, where feedback paths are broken according to the Gauss–Seidel time advancement scheme; the blocks declared by the user as circuit blocks are analyzed with "standard" simulation algorithms as defined in Section I, where feedback is fully taken into account.

Mixed mode simulators allow a smooth transistion between different levels of simulation but they also allow the designer

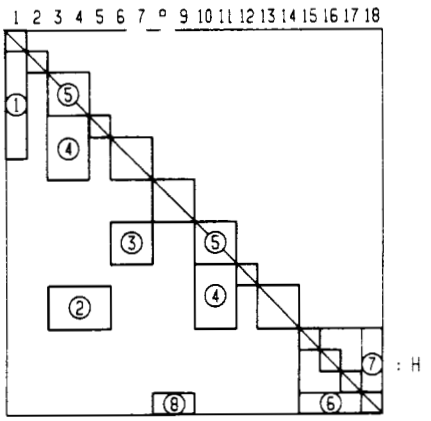


Fig. 9. Event scheduling and the BT form.

to take advantage of the time and memory savings available from higher level descriptions of blocks of the circuits. Some basic issues for mixed mode simulators are 1) how to manage the interfaces between the various levels of circuit description and simulation, and 2) the scheduling of the blocks to be processed during the analysis to exploit the inactivity or latency of part of the circuit. Since the interface problems have been discussed thoroughly elsewhere, [18], in this section we shall focus on the scheduling algorithms.

The event scheduler is the part of logic, timing, and mixed mode simulators which establishes the order in which the blocks of the circuit have to be processed. Thus we can establish an equivalence between the task of finding the *intrinsic* block structure of the dependency matrix of a given system and the basic task of the event scheduler. Event schedulers therefore play an important role in exploiting latency. The remaining of the section is devoted to the analysis of the basic tasks of an event scheduler, i.e., determining the block structure of the circuit and exploiting latency. We conclude the section with a brief description of an event scheduling algorithm.

A. Determining the Block Structure and Related Tasks

In Fig. 9, the equivalence between the basic task of the event scheduler and the task of finding the block structure of the dependency matrix can be seen by tracing the sequence of events which follow a change in input variable block "1." We assume that the system is initially in equilibrium and that all blocks have the same unit delay. We can then determine that due to the nonzeros in submatrix 1 (circled numeral) variable blocks "3–7," should be scheduled for processing at the second time step. After the second time step, due to the nonzeros in submatrices 2 and 3, variable blocks "10, 11," and "13, 14" should be scheduled for processing at the third time step. Finally, due to submatrix 4, variable block "12" is scheduled for processing at the fourth time step. Note variable blocks "2," "8, 9" and "15–18" are not scheduled.

Of course, the event schedulers used in mixed mode simulators perform more sophisticated tasks as well, but these may also be viewed in terms of the BT structure of Fig. 9. One such task is the so-called "backtracing," [14], problem of finding the cone of influence of a given variable, i.e., the set of variables in the system which must be determined before the given variable can be updated. Another such task is that of maintaining a storage-efficient hierarchical representation of the block structure for scheduling purposes. These tasks are discussed in turn later.

We illustrate in Fig. 9 the task of backtracing. Suppose we wish to know the cone of influence of the block of variables labeled "11" at the top of Fig. 9(a). We assume that the dependency matrix A is zero everywhere except on the diagonal and in the blocks explicitly identified by solid lines in the lower triangle, which we assume for simplicity to be full. Since the submatrix 5 (E_{21}) is full, block "10" must be computed along with "11." Before either of these can be computed, variable block "6, 7" must be known, due to the existence of submatrix 3. But the computation of "6, 7" must be preceded by that of "1, 3, 4." Therefore, the cone of influence of variable block "11" is "1, 3, 4, 6, 7, 10." The algorithmic embodiment of this idea is known as critical path analysis [33]. The critical path algorithm is the essential step in the "critical path timing simulation" mentioned in the list above. This type of simulation, in which the circuit blocks have delay attributes, but no other logical or algebraic properties, is an extremely important tool in the timing verification of large digital systems [14].

As an example of the use of this algorithm, suppose that a critical path timing analysis of the system characterized by the dependency matrix A of Fig. 9 has indicated that output variable block "11" was on a critical path (i.e., a path of maximum delay). In order to make the system faster, it is then desirable to simulate only the subcircuits which contribute to the critical path delay. But these subcircuits are precisely those whose variable blocks are in the cone of influence of variable block "11." Circuit simulation of *only* the cone of influence can be accomplished by arming the event scheduler with this information. The scheduler can then avoid the scheduling of any circuit block which might be active, but not on the critical path.

The use of hierarchy by an event scheduler can be discussed in terms of Fig. 9 as well. Note that subcircuit H in the lower right corner has a dependency matrix which has a BBDF substructure. Thus the sub-subcircuits represented by the variable blocks "8, 9, and 10" are connected to the external world only to variable block "18," (through submatrix 8). Thus variable blocks "15, 16, and 17" will never be scheduled until after "18" has been completed, as indicated by the existence of submatrix 7 to the right of these blocks in the overall dependency matrix.

B. Exploiting Latency

Exploiting latency is a key point to make the analysis of very-large-scale circuits economically feasible. Latency is traditionally exploited in logic simulators where event driven simulation is widely used [13], [15]. Only recently, with the advent of mixed mode simulators, similar techniques have been introduced at the electrical (timing and circuit) levels. "Bypass" schemes were used previously blocks of the circuit. These schemes required checking each block for inactivity. The overhead associated with checking for inactivity can become a substantial fraction of the total analysis run. Selective trace algorithms implemented in event schedulers avoid checking for inactivity by identifying and scheduling for analysis only the blocks which can be affected by a change in an input which is exercised during simulation. Once the effect of the change in the input disappears in any sequence of blocks, that sequence need not be traced further.

Most of the efficiencies that have been claimed in the name of latency can be described in terms of Fig. 9. Since variable blocks "2, 8, 9, 15–18" are not scheduled in response to a change in input block "1," these blocks may be said to be latent. Therefore, no computations on these blocks need to be done. In this case, these blocks had no topological connection to the inputs which changed.

As mentioned above, a similar situation occurs when it is desired to analyse only a critical path and not the remainder of a network. However, in this case the devices not analysed would actually be active if they were considered for processing, so they cannot be said to be latent.

Related to the "topological" aspect of latency discussed above, there is a dynamic aspect. A long chain of logic gates illustrates this point, in the sense that the responses to a step at the beginning of the chain propagates like a wave motion down the chain. Gates ahead of or behind the wave front, may be said to be latent, even though they are all topologically connected to the input. The degree of latency depends on the "length" of the shortest path (in the graph of the dependency matrix) between a given gate and the set of active gates. Because of the dynamic aspect, each gate in the shortest path has an inherent delay associated with it. In unit delay simulation the event scheduler schedules all gates on a path of length 1 from an active gate. Thus if the dependency matrix is full, no gates can be latent. In nominal delay simulation, gates have variable delay attributes. In this case short paths can have long delays and *vice versa*.

Therefore, when the blocks considered are analyzed at the electrical level, the dynamic behavior of the internal variables of the blocks have to be considered before declaring a block latent. For example, the inputs of a given block may be quiescent, but if the internal variables of the block are still changing from time step to time step, the block cannot be said to be latent, and must be processed until the internal variables also become quiescent [17].

C. An Event Scheduling Algorithm for Timing Analysis

Timing simulation algorithms of the type described in Section V require not only the identification of the intrinsic block structure of the circuit as discussed above, but the identification and elimination of feedback loops as well. This additional complication makes the discussion of an event scheduler for timing simulation particularly interesting. Moreover, the event scheduler of a mixed mode simulator has the same basic structure, the only difference being the need to analyze differently electrical and logic blocks [14]. The context of the algorithm taken from [14], is SPLICE type timing simulation with a one-step Gauss–Seidel time advancement scheme. In the algorithm we shall use $v_{i,n+1}$ to stand for the ith node voltage at time t_{n+1}. Also we use $\hat{t}_i$ to stand for the time at which the ith node voltage was last processed. We shall refer to the nodes which have nonzeros in the ith column of the dependency matrix as the fanouts of node i.

Procedure EVENT:

```
WHILE there are nodes scheduled to be processed at t_{n+1},
  BEGIN
  get the next scheduled node, i;
  IF(t̂_i = t_{n+1})
    BEGIN
    schedule node i for processing at t_{n+2};
    END
  ELSE
    BEGIN
    process node i at t_{n+1};
    set t̂_i = t_{n+1};
    IF(v_{i,n+1} ≠ v_{i,n})
      BEGIN
      schedule node i for processing at time t_{n+2};
      schedule all fanouts of node i for processing at time t_{n+1};
      END
    END
  END
END.
```

Note that this algorithm is an example of unit-delay scheduling, since nodes are scheduled only at t_{n+1} or t_{n+2}.

The execution of this algorithm can be explained by describing the three principal cases. First, consider the case in which neither of the IF conditions are satisfied. In this case, node i does not cause any other nodes to be scheduled for later processing, so that the fanouts of this node which have not already been scheduled, remain latent. For example, suppose that in the system of Fig. 9, variable blocks "1, 3, 4, 5, 6" have already been processed and we are currently processing variable block "7." Since by assumption these variables are not changing, the rest of the system will remain latent in this case.

In the second case, we assume that the second IF condition is satisfied but not the first. In the example described above, since variable block "7" is changing in this case, it will cause itself to be scheduled at time t_{n+2} and will cause variable blocks "10, 11" to be scheduled for processing at the current time step t_{n+1}.

In the third case we assume that the first IF condition is satisfied. In this case, a variable is scheduled which has already been processed at the current time step. Note that the scheduler schedules this node at the next time step rather than at the current time step. This is the specific mechanism with which the SPLICE Gauss–Seidel time advancement scheme limits itself to a single relaxation sweep through the overall network.

Other event schedulers work with an arbitrary (sometimes integer) scheduling time. Such schedulers are helpful in identifying timing races and other hazards. The basic idea is to compare scheduled times for the inputs and output of a given gate with the gate delay and or clock constraints.

VII. Conclusions

We have presented a survey of recent literature on third generation circuit simulation. We have chosen large scale decomposition as a theme for the paper and we have emphasized the role of intrinsic and derived triangular matrix forms which are exploited in various and disparate ways by the algorithms employed by contemporary LSI and VLSI scale simulators. We have classified decomposition techniques into two categories: tearing decomposition and temporal decomposition. These methods differ in the way feedback between blocks of the decomposition is treated. Tearing methods take the feedback fully into account, while temporal decomposition methods achieve decomposition by cutting feedback paths and then performing an inexpensive simulation which approximates the effects that feedback has on the blocks of the decomposition. This classification has been carried through all levels of circuit simulation algorithms: linear equation level, nonlinear equation level, ordinary differential equation level. In addition we have discussed methods for identifying an appropriate topological decomposition when this decomposition is not functional, i.e., specified in advance by the designer.

We have shown that the more revolutionary third generation simulators achieve their scale advantage over standard second generation simulators by relaxing the numerical and stability requirements which gave standard algorithms their generality and robustness. This deficiency is made up for by restricting the class of applicable circuits, e.g., to MOS circuits with node to ground capacitance at every node (MOTIS). We have finally described the use of event scheduling algorithms to identify the intrinsic block structure of large scale circuits and to exploit latency.

With the advent of the VLSI era, it will soon be possible to implement decomposition algorithms in hardware to gain orders of magnitude in execution time of simulation algorithms. We forecast that fourth generation simulators will be totally hardwired in VLSI chips. For this to occur, further research on the relationships between algorithms and chip architecture will be necessary.

Acknowledgment

The authors acknowledge the benefits of informative and stimulating discussions with R. K. Brayton, A. R. Newton, A. E. Ruehli, M. R. Lightner and E. Lelarasmee.

References

[1] L. W. Nagel, "SPICE2: A computer program to simulate semiconductor circuits," ERL Memo ERL-M520, University of California, Berkeley, May 1975.

[2] a) W. T. Weeks, A. J. Jiminez, G. W. Mahoney, D. Mehta, H. Qassemzadeh, and T. R. Scott, "Algorithms for ASTAP—A network analysis program," *IEEE Trans. Circuit Theory*, vol. CT-20, pp. 628–634, Nov. 1973.
b) G. D. Hachtel, R. K. Brayton, and F. G. Gustavson, "The sparse tableau approach to network analysis and design," *IEEE Trans. Circuit Theory*, vol. CT-18, pp. 101–113, Jan. 1971.

[3] C. Ho, A. E. Ruehli, and P. Brennan, "The modified nodal approach to network analysis," *IEEE Trans. Circuits Syst.*, vol. CAS-25, pp. 504–509. June 1975.

[4] a) B. R. Chawla, H. K. Gummel, and P. Kozak, "MOTIS—An MOS timing simulator," *IEEE Trans. Circuits Syst.*, vol. CAS-22, pp. 901–909, Dec. 1975.
b) H. N. Nham and A. K. Bose, "A multiple delay simulator for MOS LSI circuits," in *Proc. 17th Design Automation Conf.* (Minneapolis, MN), June 23–25, 1980.
c) V. D. Agrawal *et al.*, "A mixed mode simulator," in *Proc. 17th Design Automation Conf.* (Minneapolis, MN), June 23–25, 1980.

[5] A. R. Newton, "Techniques for the simulation of large-scale integrated circuits," *IEEE Trans. Circuits Sys.*, vol. CAS-26, pp. 741–749, Sept. 1979.

[6] a) G. Arnout and H. De Man, "The use of threshold functions and Boolean-controlled network elements for macromodelling of LSI circuits," *IEEE J. Solid-State Circuits*, vol. SC-13, pp. 326–332, June 1978.
b) H. De Man *et al.*, "DIANA: Mixed mode simulator with a hardware description language for hierarchical design of VLSI," in *IEEE ICCC'80 Conf. Proc.* (Rye, NY), pp. 356–360, Oct. 1980.

[7] a) K. Sakallah and S. W. Director, "An activity-directed circuit simulation algorithm," *IEEE ICCC'80 Conf. Proc.* (Rye, NY), pp. 1032–1035, Oct. 1980.
b) A. E. Reuhli, E. Lelarasmee, and A.L.M. Sangiovanni-Vincentelli, "The waveform relaxation decoupling approach to large scale circuit simulation," to be published in Internal IBM Res. Rep., 1981.
c) E. Lelarasmee, private communication, Nov. 1980.

[8] P. M. Trouborst and J.A.G. Jess, "Macromodelling by systematic code reduction," in *IEEE ICCC'80 Conf. Proc.* (Rye, NY), pp. 337–340, Oct. 1980.

[9] W.M.G. van Bokhoven, "Macromodelling and simulation of mixed analog–digital networks by a piecewise-linear systems approach," in *IEEE ICCC'80 Conf. Proc.* (Rye, NY), pp. 361–365, Oct. 1980.

[10] I. N. Hajj, "Sparsity considerations in network solution by tearing," *IEEE Trans. Circuits Syst.*, vol. CAS-27, pp. 357–366, May 1980.

[11] P. Yang, I. N. Hajj, and T. N. Trick, "Slate: A circuit simulation program with latency exploitation and node tearing,"

[12] S. A. Szygenda and E. W. Thompson, "Digital logic simulation in a time-based, table-driven environment. Part 1. Design verification," *Computer*, pp. 24–36, Mar. 1975.

[13] E. G. Ulrich, "Time sequenced logical simulation based on circuit delay and selective tracing of active network path," in *Proc. ACM 20th Nat. Conf.*, pp. 437–448, 1965.

[14] A. R. Newton, "Timing, logic and mixed mode simulation for large MOS integrated circuits," in *Proc. NATO Advanced Study Institute on Computer Design Aids for VLSI Circuits* (Urbino, Italy), Aug. 1980.

[15] E. G. Ulrich, "Exclusive simulation of activity in digital networks," in *Commun. ACM*, vol. 12, no. 2, pp. 102–110, Feb. 1969.

[16] a) A. Westerberg and T. Berna, "Decomposition of very large-scale Newton–Raphson based flowsheeting problems," *Comput. Chem. Eng.*, vol. 2, no. 1, pp. 61–63, 1978.
b) A. Westerberg *et al.*, *Process Flowsheeting*. Cambridge, England: Cambridge, Univ. Press, 1979.

[17] a) N. B. Rabbat, A. L. Sangiovanni-Vincentelli, and H. Y. Hsieh, "A multilevel Newton algorithm with macromodeling and latency for analysis of large-scale nonlinear networks in the time domain," *IEEE Trans. Circuits Syst.*, vol. CAS-26, no. 9, Sept. 1979.
b) A. Sangiovanni-Vincentelli and N.B.G. Rabbat, "Techniques for the time domain analysis of VLSI circuits," *Proc. Inst. Elec. Eng.*, vol. 127, part G, pp. 292–301, 1980.

[18] H. De Man, "Computer aided design for integrated circuits: Trying to bridge the gap," *IEEE J. Solid-State Circuits*, vol. SC-14, pp. 613–621, June 1979.

[19] J. M. Ortega and W. C. Rheinboldt, *Iterative Solution of Nonlinear Equations in Several Variables*. New York: Academic Press, 1970.

[20] C. W. Gear, "The automatic integration of stiff ODE's," in *Proc. IFIPS Congr.*, pp. A81–A85, 1968.

[21] R. K. Brayton, F. Gustavson, and G. D. Hachtel, "A new efficient algorithm for solving differential-algebraic systems using implicit backward differentiation formulas," in *Proc. IEEE*, vol. 60, pp. 98–108, Jan. 1972.

[22] Gustavson, F. G., "Some basic techniques for solving sparse systems of linear equations," in *Sparse Matrices and Their Applications*, D. J. Rose and R. A. Willoughby, Eds. New York: Plenum Press, 1971.

[23] A. Westerberg, "La Scala—A programming package for the solution of linear systems," in *Proc. SIAM Nat. Meet.* (Knoxville, TN), 1979.

[24] J. Sherman and W. J. Morrison, "Adjustment of an inverse matrix corresponding to changes in the elements of a given column or row of the original matrix," *Annu. Math. Statist.*, vol. 20, p. 621, 1949.

[25] a) J. A. Geroge and J.W.H. Liu, "A quotient graph model for symmetric factorization," in *SIAM Sparse Matrix Proc. 1978*, I. S. Duff and G. W. Stewart, Eds., pp. 154–175, 1978.
b) J. A. George, J. Liu, and E. Ng, *User Guide for SPARSEPAK*, Dep. of Computer Science, Univ. Waterloo, Waterloo, Ont. Canada, June 1979.
c) J. A. Geroge and J.W.H. Liu, "A fast implementation of the minimum degree algorithm using quotient graphs," *ACM TOMS*, vol. 6, no. 3, pp. 337–358, Sept. 1980.

[26] a) J. A. George, and F. G. Gustavson, "A new proof on permut-

ing to block triangular form," IBM RC Rep. 8238, 10 pp., Apr. 1980.
b) J. A. George, "Nested dissection of a regular finite element mesh," *SIAM J. Numer. Anal.*, vol. 11, pp. 345-363, 1974.
c) ——, "On block elimination for sparse linear systems," *SIAM J. Numer. Anal.*, pp. 585-603, 1974.
[27] I. N. Hajj, "Solution of interconnected subsystems," *Electron. Lett.*, vol. 13, no. 3, pp. 78-79, Feb. 1977.
[28] R. K. Brayton and R. Spence, *Sensitivity and Optimization*. New York: Elsevier Scientific Publ., 1980.
[29] A. Sangiovanni-Vincentelli, L. K. Chen, and L. O. Chua, "A new tearing approach—Node-tearing nodal analysis," in *Proc. 1977 IEEE Int. Symp. Circuits and Systems* (Phoenix, AZ), pp. 143-147, Apr. 1977.
[30] ——, "An efficient heuristic cluster algorithm for tearing large-scale networks," *IEEE Trans. Circuits Syst.*, vol. CAS-24, pp. 709-717, Dec. 1977.
[31] C. J. Pottle, "A parallel processing architecture for power system load flow computations," in *IEEE ICCC'80 Conf. Proc.* (Rye, NY), pp. 801-805, Oct. 1980.
[32] a) F. F. Wu, "Solution of large-scale networks by tearing," *IEEE Trans. Circuits Syst.*, vol. CAS-23, pp. 706-713, Dec. 1976.
b) F. F. Wu, Diakoptic network analysis," in *Proc. IEEE 1975 Power Industry Computer Applications (PICA)* (New Orleans, LA), pp. 364-371, June 1975.
[33] E. Lawler, *Combinatorial Optimization, Networks and Matroids*. New York: Holt, Rinehart and Winston, 1976.
[34] G. D. Hachtel *et al.*, "Semiconductor analysis using finite elements—Part I: Computational aspects," *IBM J. Res. Dev.*, July, 1981.
[35] E. Buturla *et al.*, "Finite element modeling in 2 or 3 space dimensions and time," *IBM J. Res. Dev.*, July 1981.
[36] 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.
[37] J. P. Roth, *Computer Logic, Testing and Verification*. Potomac, MD: Computer Science Press., 1980.
[38] E. Hellerman and D. Rarick, "Reinversion with the preassigned pivot procedure," *Math. Program.*, vol. 1, pp. 195-216, 1971.
[39] a) A. K. Kevorkian and J. Snoek, "Decomposition in large scale systems: Theory and applications in solving large sets of nonlinear simultaneous equations," in *Decomposition of Large Scale Problems*, D. M. Himmemblau, Ed. Amsterdam, The Netherlands: North Holland, 1973.
b) A. K. Kevorkian, "A decompositional algorithm for the solution of large systems of linear algebraic equations," in *Proc. 1975 IEEE Int. Symp. Circuits Systems*, pp. 116-120, 1975.
[40] ——, "On bordered triangular or lower N forms of an irreducible matrix," *IEEE Trans. Circuits Syst.*, vol. CAS-23, pp. 621-624, 1976.
[41] L. K. Cheung and E. S. Kuh, "The bordered triangular matrix and minimum essential set of digraph," *IEEE Trans. Circuits Syst.*, vol. CAS-21, pp. 633-639, 1974.
[42] G. Guardabassi, "A note on minimal essential sets," *IEEE Trans. Circuit Theory*, vol. CT-18, pp. 557-560, 1971.
[43] a) G. W. Smith and R. B. Walford, "The identification of a minimal feedback vertex set of a directed graph," *IEEE Trans. Circuits Syst.*, vol. CAS-22, pp. 9-15, 1975.
b) H. Y. Chang, G. W. Smith, Jr., and R. B. Walford, "LAMP: System description," *Bell Syst. Tech. J.*, vol. 53, pp. 1431-1449, Oct. 1974.
[44] J. Katzenelson, "An algorithm for solving nonlinear resistive networks," *Bell Syst. Tech. J.*, vol. 44, pp. 1605-1620, 1965.
[45] T. Fujisawa and E. S. Kuh, "Piecewise-linear theory of nonlinear networks," *SIAM J. Appl. Math.*, vol. 22, no. 2, Mar. 1972.
[46] E. Hellerman and D. Rarick, "The partitioned preassigned pivot procedure (P4)," in *Sparse Matrices and Their Applications*, D. J. Rose and R. A. Willoughby, Eds. New York: Plenum Press, 1972.
[47] A. Sangiovanni-Vincentelli and T. A. Bickart, "Bipartite graphs and an optimal bordered triangular form of a matrix," *IEEE Trans. Circuits Syst.*, vol. CAS-26, no. 10, pp. 880-890, Oct. 1979.
[48] R. K. Brayton, F. G. Gustavson, and R. A. Willoughby, "Some results on sparse matrices," *Math. Comput.*, vol. 24, no. 112, 1970.
[49] M. F. Moad, "A sequential method of network analysis," *IEEE Trans. Circuit Theory*, vol. CT-17, pp. 99-104, Feb. 1970.
[50] G. Guardabassi and A. Sangiovanni-Vincentelli, "A two-level algorithm by tearing," *IEEE Trans. Circuits Syst.*, vol. CAS-23, pp. 783-791, 1976.
[51] J. D. Crawford, M. Y. Hsueh, A. R. Newton, and D. O. Pederson, *MOTIS-C User's Guide*, Electronics Research Laboratory, Univ. California, Berkeley, June 1978.
[52] G. De Micheli, A. Sangiovanni-Vincentelli, and A. R. Newton, "New algorithms for timing analysis of large circuits," in *Proc. IEEE Int. Symp. Circuits and Systems*, 1980.
[53] G. De Micheli and A. Sangiovanni-Vincentelli, "Numerical properties of algorithms for analysis of MOS VLSI circuits," in *Proc. European Conf. Circuit Theory and Systems*, Aug. 1981.
[54] W. Kahan, Univ. California, Berkeley, Private Communication.
[55] H. DeMan, G. Arnout, and P. Reyneart, "Mixed mode circuit simulation techniques and their implementation in DIANA," in *Proc. NATO Advanced Study Institute, SOGESTA* (Urbino, Italy), Aug. 1980.
[56] A. E. Ruehli, A. Sangiovanni-Vincentelli, and N.B.G. Rabbat, "Time analysis of large scale circuits using one-way macromodels," in *Proc. IEEE Int. Symp. Circuits and Systems*, pp. 766-770, 1980.
[57] R. E. Tarjan, "Depth first search and linear graph algorithms," *SIAM J. Comput.*, vol. 1, pp. 146-160, 1972.
[58] F. G. Gustavson, "Finding the lower triangular form of a sparse matrix," in *Sparse Matrix Computations*. J. Bunch and D. Rose, Eds. New York: Academic Press, 1976.
[59] A. George, "An automatic one-way nested dissection algorithm for irregular finite element problems." *SIAM J. Numer. Anal.*, vol. 17, pp. 740-751, 1980.