Waveform-relaxation-based circuit simulation on the Victor V256 parallel processor

by T. A. Johnson
D. J. Zukowski

Present-day circuit-analysis tools permit designers to verify performance for circuits consisting of up to 10 000 transistors. However, current designs often exceed several tens of thousands and even hundreds of thousands of transistors. The gap between the number of transistors that can be simulated and the number per design inhibits proper analysis prior to manufacturing, yet incomplete analysis often overlooks design flaws and forces redesign, resulting in increased costs and longer development times. This gap is expected to widen in the foreseeable future. To help close the ever-increasing simulation/design gap, we have developed an experimental parallel circuit simulator, WR_V256, for the Victor V256 distributed-memory parallel processor. WR_V256 has been used to analyze circuits from fewer than 300 to more than 180 000 MOSFETs. WR_V256 was originally based on a Gauss–Seidel relaxation algorithm, which was later replaced with a bounded-chaotic one in order to achieve good parallel speedups for a wider variety of circuits. At this time, speedups of up to 190 have been observed for large circuits.

1. Introduction

The 1980s have witnessed an explosion in the field of circuit design. During that decade, the size and complexity of integrated circuits have increased so rapidly that many of the supporting tools that triggered this explosion have fallen far behind. Such is the case with circuit simulation. Most present-day circuit-analysis tools such as SPICE [1] and ASTAP [2] are limited in their ability to achieve reasonable turnaround times for large jobs and to cope with the storage requirements for those jobs. Typically, such conventional circuit-analysis tools are effective for analyses of not more than a few thousand transistors. Waveform relaxation, as implemented in the RELAX [3] and TOGGLE [4] programs, has increased the size of digital MOS circuits that can be analyzed efficiently to a

few tens of thousands of FETs. However, even these tools are not sufficient to keep pace with the rapidly expanding needs of circuit designers. To adequately and accurately analyze the complex VLSI chips being developed today, circuit simulation must be capable of handling many tens of thousands of FETs in one analysis and must analyze such circuits rapidly.

Several approaches have been taken to analyze large circuits on serial computers. One approach is to manually partition large circuits into many smaller ones and analyze each independently of the rest. Manual partitioning techniques are time-consuming and often introduce errors by imposing artificial interfaces that cause interactions among partitions to be lost. Mixed-mode simulation, another approach, yields faster analyses by simulating some parts of a circuit less accurately than others. The effectiveness of this technique is limited for high-performance designs because of a potential loss of overall accuracy. Additionally, user intervention is often required, to specify critical paths of a circuit. This limits feasibility for very large circuits. The use of vector processors to speed up circuit simulation is another alternative. However, efficient vectorization of circuit simulation is constrained by inadequate vector lengths and by data dependencies. Typical results [5] show limited speed improvement.

Another alternative for analyzing large circuits is to take advantage of parallel machines. One approach is to parallelize a conventional circuit simulator such as SPICE or ASTAP [6–13]. For small- to medium-sized problems of about 1000 transistors, coefficient evaluation ("filling the matrix") often consumes up to 80% of the total run time, with most of the remaining time spent in matrix solution [14]. Parallelizing coefficient evaluation has been investigated by several authors, who report varying degrees of success [6–10]. (Clearly, there is a limit to the effectiveness of such an approach: If coefficient evaluation does indeed consume 80% of an analysis, and if it is possible to sufficiently parallelize this work—i.e., the time needed approaches zero—the overall speedup is limited to a factor of 5.) As circuit size increases, the size of the solution matrix also increases and, in contrast to the case with coefficient evaluation, the time required for matrix solution grows nonlinearly with matrix size. Therefore, for large circuits, matrix solution dominates run time [15]. Possibilities for exploiting more parallelism include parallelizing matrix solution, local truncation-error estimation, and convergence testing, in addition to coefficient evaluation. Typical speedups (serial run time/parallel run time) from parallelizing all of these steps for small to medium problems are in the range of 2 to 4 [9–13]. In addition, published results, such as those of [7], show that very little improvement is gained with more than about eight processors.

Since results from parallelizing standard SPICE/ASTAP algorithms have been so disappointing, a totally different simulation algorithm has been investigated in order to increase the amount of parallelism. Waveform relaxation [16, 17], a newer analysis technique with more inherent parallelism, has received considerable attention over the last decade. While the application of waveform relaxation is currently limited to digital MOS circuits, many large designs fall into this category. Waveform relaxation is capable of automatically partitioning large circuits into many smaller, nearly independent subcircuits. With this approach, many subcircuits may be simulated in parallel, with voltage waveforms exchanged at their interconnections, until global convergence is reached. The waveform-relaxation-based algorithm has been at the center of much research on parallel circuit simulators [14, 18–25]. A more detailed description of waveform-relaxation algorithms can be found in Section 2.

Though many efforts have been made to parallelize circuit simulation, most published work has been limited to machines with a low degree of parallelism (e.g., 8–32 processing elements) and relatively small circuits (about 1000 MOSFETs). It is not clear how well results of such studies can be generalized to larger circuits analyzed on more highly parallel machines. The WR_V256 project extends previous research to provide a better understanding of the impact of larger circuits and more highly parallel machines on achievable parallel performance.

WR_V256, a waveform-relaxation-based parallel circuit simulator, was developed for the Victor V256 machine, a distributed-memory parallel processor, discussed briefly in Section 3 and described in more detail elsewhere in this journal [26]. The Victor V256 machine was used because it offered stable hardware with a high degree of parallelism. TOGGLE, an existing serial waveform-relaxation program described in Section 4, was adopted for this machine. A goal of the WR_V256 project was to achieve good parallel efficiency on existing Victor V256 hardware while changing as little as possible in the TOGGLE program. However, some new features were added to improve the behavior of circuits exhibiting poor parallel efficiencies. In particular, a bounded-chaotic relaxation, as described at the end of Section 4, was implemented to achieve better parallel execution for smaller or more irregular jobs.

This work has demonstrated that very simple techniques can be used to port a serial waveform-relaxation-based circuit simulator to a distributed-memory parallel processor, and that they are capable of achieving high parallel efficiency for many circuits. These simple techniques are also discussed in Section 4. In addition, this work shows that the size of circuits does affect parallel efficiency results. For large circuits, a speedup of nearly 190 has been demonstrated for the Victor V256 machine.

These conclusions are described more fully in Section 5, which also introduces the concept of the parallel signature of a circuit and how it can be used to explain some observed performance results. Some closing remarks and suggestions for future research are included in the final section of the paper.

2. Waveform relaxation

• *Methods*

Conventional circuit simulation tools such as SPICE and ASTAP apply Kirchoff's current and voltage laws to the topology and "branch relations" of a circuit to derive a set of nonlinear differential equations describing the circuit behavior. These equations are then transformed into a set of nonlinear algebraic equations through the use of an integration method. In general, such integration methods are of the implicit multistep type. Among the most commonly used techniques are the backward-Euler, trapezoidal-rule, and Gear's methods. The resulting algebraic equations are then linearized and solved over a user-specified analysis interval (from the "start time" to the "stop time"). Linearization is performed by application of the iterative Newton–Raphson method. Once linearized, the set of algebraic equations is usually solved by LU decomposition with sparse-matrix techniques. At the start of the circuit analysis, all of the network equations are solved iteratively at the first time point until the difference between two successive iterations is sufficiently small—i.e., the solution converges. Once convergence is achieved at the first time point, a time step is calculated to determine the next time point. The size of the time step is determined by the difficulty encountered in solving the circuit at the current (i.e., the first) time point. The iterative solution process is repeated at the next time point and at all successive time points, until the stop time is reached. With such a scheme, once the stop time is reached, the circuit waveforms (the calculated voltages at all time points for all voltage nodes) are known for the entire analysis interval, and the simulation is complete.

Although this technique produces an accurate prediction of circuit behavior, it suffers several performance problems as the size of the analyzed circuits increases. First, since all circuit equations are solved at the same time points, and each time point is determined on the basis of the difficulty in converging at the previous time point, the entire system of equations is solved using time steps determined by the most difficult set of equations. This causes the entire system of equations to be solved repeatedly even when only a small part of the circuit is converging slowly. Second, as the size of circuits grows, the time required to solve the set of equations grows nonlinearly. The growth factor is generally acknowledged to be proportional to N^{α}, where N is the order of the matrix of linear equations (N is generally proportional to the number of voltage nodes contained in the circuit), and α is between 1.4 and 1.6 [15]. So, not only are conventional methods constrained to solve the circuit equations repeatedly on the basis of the most slowly converging part of the circuit, but also the time required to solve those equations grows dramatically with circuit size.

Unlike conventional methods, waveform-relaxation methods do not attempt to solve the complete set of equations for a circuit simultaneously. Instead, large digital MOS circuits are algorithmically partitioned into interconnected subcircuits. These subcircuits are solved iteratively. For ease of exposition, the Gauss–Jacobi (GJ) relaxation algorithm is described here; the Gauss–Seidel (GS) and bounded-chaotic (BC) algorithms used in this project are described in later subsections. In the GJ algorithm, the subcircuits are solved independently, with the methods described above, first for a dc solution (to use as the initial set of voltage-node waveforms) and then for the transient solutions. The following discussion is limited to the transient solution, since it dominates the analysis. The first time the subcircuits are solved for the entire analysis interval is called the first waveform-relaxation (WR) iteration. The voltage-node waveforms calculated during the first WR iteration are used as the inputs for the next WR iteration, during which a new set of waveform solutions is calculated. Additional WR iterations are performed if the difference in voltage-node waveforms between successive iterations is outside some specified error bound. Otherwise convergence has been achieved.

Although there are several methods of defining subcircuit partitions, two of the more popular ones are pointwise partitioning and block partitioning. Pointwise partitioning simply breaks the circuit at each node, generating subcircuits with only one node. Block partitioning groups one or more circuit nodes into a single subcircuit, on the basis of the strength of coupling provided by the circuit elements that connect them. Generally, block partitioning leads to faster convergence, while pointwise partitioning offers potential for greater parallelism. An investigation of the effects of these partitioning approaches for parallel machines is discussed in [22].

Waveform relaxation has several advantages over conventional methods. First, when a large network is partitioned into a set of subcircuits, the size of each subcircuit that must be solved does not grow appreciably with the size of the problem analyzed. Since the size of each subcircuit remains approximately the same, the time to solve each subcircuit matrix also remains the same. Instead, the number of subcircuits grows roughly linearly with the size of the circuit. This results in a nearly linear growth in solution time with circuit size rather than the nonlinear growth seen with conventional methods [16, 18]. Second, each of the subcircuits can be solved using a

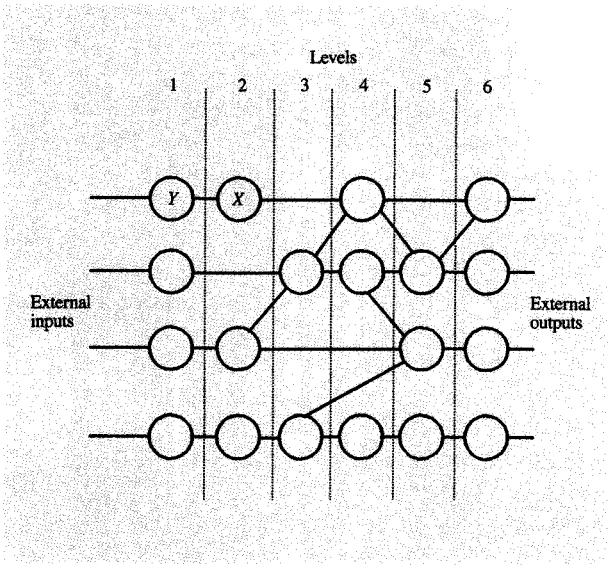

Figure 1

Example of subcircuit leveling.

sequence of time steps appropriate for its own convergence behavior. Those subcircuits that converge rapidly are not constrained to use time steps determined by others that converge more slowly. Third, and most important for a parallel implementation, since each subcircuit is solved relatively independently, the subcircuits can be analyzed in parallel.

- *Algorithms*

Within the general class of solution methods referred to as waveform relaxation, there exist several distinct relaxation algorithms, two of which are described here. (The description of the BC algorithm is reserved until the end of Section 4.) The GJ algorithm, described earlier, can be characterized by the following equation for $\mathbf{X}(t)$, the vector of node voltages:

$$\mathbf{X}^{k+1}(t) = f[\mathbf{X}^{k}(t), \dot{\mathbf{X}}^{k}(t), \mathbf{u}(t), t], \tag{1}$$

where k is the waveform iteration number, $\mathbf{u}(t)$ is a vector of external inputs, and t is time. To compute new values for the solution vector, $\mathbf{X}(t)$, the GJ algorithm uses the results of the previous iteration.

In contrast, the GS algorithm uses a combination of results from the current and previous iterations. The GS algorithm forces an ordering of the analysis; hence some serialization among subcircuits occurs. Each subcircuit is assigned a "level" determined by its inputs. Subcircuits that depend only on external inputs are defined to be level 1. Those that depend only on outputs from level-1 subcircuits and external inputs are defined as level 2, etc. The order of analysis is based on the circuit levels; i.e., all level-i subcircuits must be analyzed before any level-i+1 subcircuit. Refer to **Figure 1** for an example of subcircuit leveling.

When cycles caused by global feedback loops occur, they are generally handled as follows. An estimate is made of the delay through each loop, and subcircuits that form a cycle having a delay shorter than some predefined threshold are merged into a single subcircuit. Longer feedback loops are broken at some point in the cycle. Input waveforms to subcircuits driven by these broken feedback paths are taken from the previous iteration. In general, broken feedback loops result in more waveform iterations to reach convergence.

The GS algorithm is shown in Equation (2):

$$\mathbf{X}_i^{k+1}(t) = f_i[\mathbf{X}_{j<i}^{k+1}(t), \mathbf{X}_{j\geq i}^{k}(t), \dot{\mathbf{X}}_{j<i}^{k+1}(t), \dot{\mathbf{X}}_{j\geq i}^{k}(t), \mathbf{u}(t), t]. \tag{2}$$

After leveling and ordering, the set of subcircuits is solved iteratively until a predetermined convergence criterion is met by every node. GS relaxation has been shown to give significant improvement in the rate of waveform convergence [16], but it restricts the degree of parallelism. At any level, only those subcircuits having all inputs from earlier levels computed for the current iteration can be analyzed in parallel. Analysis of other subcircuits awaiting computation of their inputs must be delayed.

3. Victor V256 parallel processor

- *Hardware description*

The Victor V256 machine is a distributed-memory parallel processor that consists of 256 processing elements (PEs). Each of the PEs contains four megabytes (MB) of storage [one gigabyte (GB) in all]. The PEs are configured in a two-dimensional mesh (16 × 16) and can be grouped into four arbitrary partitions of contiguous PEs. Victor also includes 10 GB of mass storage, distributed among 16 disk processors. Access to Victor V256 is provided by four hosts (chosen from among DOS-based PC/AT® or PS/2®, or AIX®-based RISC System/6000™) available to users, with one typically allocated to each partition. There is also one "super" host that can be used to reset and initialize the entire system. The parallel circuit simulator application has been run with partition sizes of 64, 128, and 256 PEs. Details of the Victor hardware are described in [26].

- *WR_V256 software environment*

While Victor now provides several languages and environments, the WR_V256 project was initiated with one of the earlier ones, namely Version 2.0 of the Parallel C compiler from 3L Ltd. (3LC) [27]. The 3LC tool set consists of a C compiler with library functions to handle explicit parallel constructs, a linker, a configurer, and a host server. The compiler provides support for multiple threads, message-passing communication primitives, and

semaphores. The host server, a program that runs on a Victor host, loads each PE with an appropriate task and handles all host I/O requests.

The message-based communication mechanism implemented, in software, for the parallel circuit simulation work provides all host-to-PE, PE-to-host, and PE-to-PE communication. In addition, a restricted broadcast mechanism has been added that allows the host to efficiently broadcast a message to all Victor PEs. PE-initiated broadcasts were not implemented.

To guarantee that all jobs that fit in storage finish, the communication mechanism was designed to be deadlock-free. Briefly, deadlock-free routing (how messages are sent from source PE to destination PE) can be accomplished in a store-and-forward network, such as that supported by Victor, as long as no loops are possible in message transit and all messages are eventually consumed at their destinations [28], thus freeing any routing resources (e.g., buffers) that they hold. A noncyclic router was developed that incorporates the idea of planes of travel (virtual communication planes) and allocates buffers uniquely among the planes. Only two planes are needed for a 2D mesh, and each plane has two degrees of freedom for travel; e.g., plane A allows travel north and east, and plane B allows travel south and west. By connecting the planes with a unidirectional bridge (i.e., buffer) from plane A to plane B, all possible combinations of travel are provided in a noncyclic fashion (**Figure 2**). The number of router buffers required depends on the number of planes and travel directions, hence remaining constant as the mesh grows. A more extended description of this type of router, and further generalizations, are covered in [29].

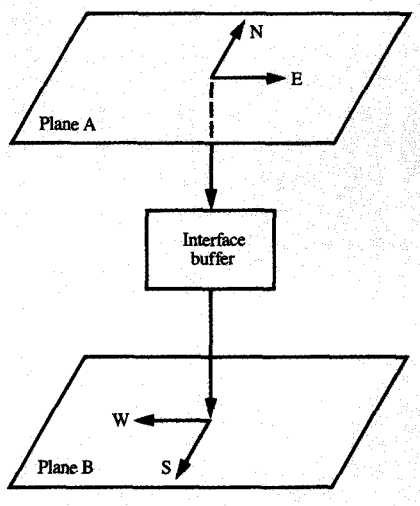

Figure 2

Planes of travel that provide deadlock-free message routing.

4. Implementation issues for a parallel circuit simulator

The TOGGLE program is one example of a waveform-relaxation-based circuit simulator. It uses a GS algorithm to minimize analysis time for execution on serial machines. Although one goal was to change as little as possible of the serial TOGGLE program, some structural changes were needed to port TOGGLE to a distributed-memory machine. These changes include data initialization and result gathering, waveform storage and updating, allocation of work, and synchronization needed to maintain consistency among the data structures of the PEs. This section first considers general aspects of the serial implementation of the TOGGLE program. It then describes each of the changes, and concludes with a discussion of one of the parallel-execution improvements added to the basic TOGGLE structure.

- *Serial TOGGLE*

The serial TOGGLE program consists of two phases. The first phase reads a circuit description, partitions the circuit into subcircuits using the block-partitioning approach, and then orders these subcircuits as described in the subsection on algorithms. The order of subcircuits within any level is arbitrary, but all subcircuits assigned to one level must be analyzed before any subcircuit of a subsequent level. A subcircuit dispatch queue is built that observes the order of the subcircuits.

The second phase of TOGGLE actually performs the analysis. Subcircuits are analyzed in the order in which they appear in the dispatch queue. An analysis is divided into two parts. First, dc values are calculated for all subcircuits. Then, the transient behavior of all subcircuits is calculated using the dc solutions as the initial conditions. Both the dc and the transient solutions may require multiple WR iterations for the node voltages to converge. After the first WR iteration, not all subcircuits need be analyzed. If all of the internal nodes of a subcircuit have converged and the inputs have not changed, the subcircuit is considered "latent" and is not solved again unless its inputs change. An analysis ends when the computed results for all subcircuits change less than a predefined amount from one WR iteration to the next.

Subcircuits influence one another's analyses through shared node voltages. Each subcircuit is responsible for the solution of its internal nodes, but more than one subcircuit may use any node waveform as input. For GS relaxation, there are two types of input, which can be characterized as follows. Inputs that are the outputs of

subcircuits of previous levels must be calculated before they can be used by another subcircuit. Inputs that model back-coupling or feedback from nodes solved at later levels are used before they are updated for the current WR iteration. All node voltage variables are accessed by means of pointers, i.e., explicit memory addresses, forcing every node voltage waveform to have a unique address. These unique addresses are available to every subcircuit. No explicit update mechanism is needed, since all updates and references are issued to the same variable via its pointer, and the order of references is restricted by the dispatch queue.

TOGGLE was heavily optimized to run on a serial machine, and several of these optimizations caused complications for a parallel implementation. A GS relaxation method was used by TOGGLE, which is inherently much more serial than a GJ method [19]. However, the GS approach was maintained for three reasons. First, the same method had to be used in order to verify the correctness of computed results and to make performance comparisons with serial execution. Second, the GJ method uses much more memory, because waveforms for both the current and previous iteration must be stored for each circuit node. This effectively doubles the amount of memory required to store computed waveforms, thus reducing the size of circuits that can be analyzed. Third, GJ usually requires more WR iterations. Therefore, while a GJ method may have a higher parallel efficiency, the overall turnaround time for a job may be greater because of the additional work needed to complete an analysis.

- *Parallel TOGGLE*

Data initialization and result gathering

The TOGGLE program was explicitly divided into two phases, with the circuit-analysis phase being separated from the partitioner/scheduler. For typical waveform-relaxation programs, the analysis phase dominates total run time, and this dominance becomes even more significant as circuit size increases [18]. Therefore, only the actual analysis phase was parallelized. (The partitioner/scheduler can be parallelized separately.) All circuit partitioning and ordering are performed on an IBM 3090™ processor, on which TOGGLE was originally installed.

Since the first part of the TOGGLE program is executed on a 3090 system, the input files needed to describe circuits are the same as for the serial implementation. However, once partitioning and ordering are complete and the allocation of subcircuits to PEs has been decided (see below), a file is built describing the static mapping onto V256. Circuit and device data from this file are then divided among the Victor disk processors. Initialization of subcircuits on Victor PEs can then proceed in parallel, using all 16 of the Victor disks. After an analysis is complete, each of the PEs sends its solved waveforms to the Victor disk connected to it. The files are then concatenated and uploaded to the 3090 system, where the solutions may be inspected using a special viewing program.

Waveform storage and update

Although V256 has 1 GB of primary storage available to an application, that storage is evenly distributed among and private to the 256 individual PEs. Waveforms are statically assigned to PEs, so that each PE contains a copy of all waveforms needed for analysis of the subcircuits allocated to that PE. Hence, copies of the same waveform may be found on several PEs. To ensure that all copies are identical, each waveform is assigned a unique name, and a new software mechanism called the Waveform Repository is used. While all waveforms resident on a PE are still accessed by means of memory addresses, the Waveform Repository (distributed on all PEs) collects all waveform addresses on a PE into a table. This table provides a way to translate global waveform names to local PE addresses. Each PE maintains entries for only those waveforms it references.

As a waveform is changed on one PE, this change must be reflected in all external copies. Note that each waveform is solved on only one PE; therefore, one PE holds the master copy and all other copies must be explicitly updated. There are two common methods used to update copies of a variable in a distributed-memory system. The spread method is controlled by the PE generating a waveform. As a waveform is modified, messages with the new values of the waveform are sent by the originating PE to all PEs that hold copies. The gather method, on the other hand, is driven by the receiving PEs. When a waveform is to be referenced (read), a message requesting an updated version is sent to the PE holding the master copy. That PE then either sends its current version, which may be refused at the receiver if the waveform is the same version as the current copy, or it waits until its version is updated. The spread method is more efficient in communication, because it does not require the request messages that the gather method does. However, it is more memory-intensive, since there must be enough storage for all externally generated inputs on every PE. The gather method need store inputs for only the subcircuit currently being analyzed, though it likely would fetch inputs for the next few subcircuits in order to minimize waits. The waveform-relaxation algorithm guarantees that all computed waveforms from one WR iteration will be used during either the current WR iteration or the next. Therefore, since all data transmitted by the spread method are used and no unnecessary request

messages are transmitted, the spread method was used for this application.

Allocation of work

To achieve high parallel efficiency, special attention must be paid to the distribution of work across PEs in the network. Inefficient communication patterns can badly degrade parallel performance. Communication of data itself may take a significant amount of time. During this time, depending on the relaxation algorithm, the receiving PE may have to wait for updated inputs. In addition, communication among PEs is performed by several processes on each PE. These communication processes compete with the analysis process for a share of available PE cycles.

Nonuniform load balance also degrades parallel performance. Good load balancing is especially difficult to achieve for circuit simulation. In circuit-analysis problems, the amount of time taken per subcircuit per WR iteration depends on several factors. First, the size of a subcircuit influences its analysis time. Second, the dynamic switching activity of a subcircuit directly affects the amount of time required to solve it. The subcircuits that are electrically active are determined by the circuit topology and the external input waveforms; those subcircuits that are active require more matrix solutions per WR iteration than those that are relatively inactive. Third, latent subcircuits may not be analyzed at all for a particular WR iteration, though they may be analyzed again in subsequent iterations.

At this stage of the project, subcircuits are statically assigned to the PEs. While dynamic load balancing works well for many applications, circuit simulation poses several problems for load-balancing algorithms. In general, each subcircuit may require many tens of thousands of bytes to contain its data, which makes moving a subcircuit from PE to PE very expensive once analysis has begun. In addition, the behavior of a subcircuit may change from one WR iteration to the next, so while one PE may be a bottleneck for one iteration, it may not be for the next. There is no way to determine ahead of time which PE will be heavily loaded, and to redistribute subcircuits in advance. Finally, a dynamic load balancer would incur substantial overhead, both for bookkeeping and for increased communication.

Instead, a two-step process (described in the following subsections) was implemented to statically assign subcircuits to the PEs. Communication among PEs is reduced by generating "chains" of subcircuits. (This approach is similar to the algorithm for partitioning by element strings [30].) The workload is better balanced by assigning multiple chains to each PE.

Chaining of subcircuits How waveform data are shared among PEs is dictated by subcircuit interconnections and by the assignment of subcircuits to PEs. It is desirable to assign subcircuits that share data to the same PE. The first step of the assignment is the creation of sequential "chains" of subcircuits of the partitioned network, each chain containing at most one subcircuit from each level. **Figure 3** illustrates the creation of a chain from the example of Figure 1. A chain is built starting with a subcircuit that generates one or more of the circuit outputs (*A* in Figure 3). The next subcircuit assigned to the chain is selected from the subcircuits that drive the first subcircuit in the chain. If more than one such subcircuit is found (*B* and *C* in Figure 3), the subcircuit at the level closest to the last subcircuit assigned to the chain is selected (*B*). If two or more subcircuits at the same level provide input to the most recent subcircuit assigned to the chain, the subcircuit with the greatest number of connections to the chain is selected (subcircuit *F* is chosen rather than *G*). At each level, the goal is to maximize connectivity of the elements within a chain. The process is repeated until either all subcircuits in the current path are exhausted or a primary input is reached. Then a new output node is selected and a new chain is started. When all output nodes are exhausted, the process is begun again by dropping back one level and starting new chains with all unassigned subcircuits in that level. This is repeated until all subcircuits are assigned to some chain. This approach increases the likelihood that some signals solved by one subcircuit will be used as input to another subcircuit on the same PE, thereby helping to reduce communication among PEs.

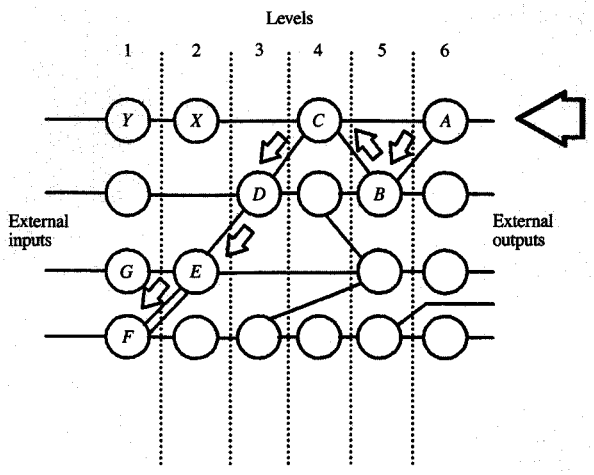

Figure 3

Chaining of subcircuits.

Sometimes it is necessary to truncate a chain before reaching a primary input. Since each Victor PE contains only 4 MB of user-addressable storage, if the subcircuits in a chain are large or the number of levels is large, it is possible to create a chain whose storage requirements cannot be satisfied by using just one Victor PE. In such cases, chains are broken into two or more sequential pieces and are assigned to separate PEs.

Grouping of subcircuit chains Next, an attempt is made to estimate the computational workload of each chain and to place several chains of subcircuits on every PE in an effort to better balance the load across PEs. As stated previously, workload estimates are often inaccurate because of the dynamic nature of circuit-analysis problems. Although the switching activity and convergence behavior of a subcircuit are unknown, its size (i.e., the order of the solution matrix) and the number of FETs contained are known in advance. Generally, subcircuits are small (approximately 10–20 nodes). Therefore, matrix coefficient evaluation (the time of which grows linearly with the number of elements) dominates run time, and the number of FETs may be used to estimate the computational requirement of a subcircuit.

Chains are assigned to PEs so that the total number of FETs allocated to each is as uniform as possible. By assigning more than one chain to a PE, not only can the workload be better balanced, but there is also a greater likelihood that a subcircuit of at least one chain will be ready for analysis at all times. I.e., should the next subcircuit of one chain need to wait for its inputs, the next subcircuit of another chain on the same PE may already have its inputs available and be ready for analysis. In addition, some of the communication time may be overlapped with another subcircuit analysis, because each PE in effect multitasks chains in its group.

Distributing the dispatch queue
Since the use of a centralized dispatch queue would have been likely to degrade performance, especially when the data must be stored statically, the dispatch queue was divided into independent queues on each PE. Thus, the mechanism that enforced GS ordering among subcircuits was eliminated. Since each Victor PE may hold subcircuits from every level, and the data dependencies that were enforced by the serial queue are lost, a data-driven approach that ensures that a subcircuit is not analyzed until all of its inputs have been updated was added. The scheduler was modified to maintain two queues: one for subcircuits whose inputs have been updated, and another for those subcircuits awaiting updated inputs.

Control-flow synchronization
Some synchronization is needed to ensure that data from one of the phases of execution do not overwrite those from another, since several copies of the node-voltage waveforms are present in the Victor system. The three phases of execution are the data structure initialization, the dc solution, and the transient solution. Should one PE finish its initialization, begin its dc solution, and send a solved waveform to a slower PE, and should that slower PE initialize the waveform copy after the update has been received, the solved waveform would be lost. A very simple synchronization mechanism that restricts a PE from entering a new phase before all PEs are finished with the current phase was introduced. As a PE finishes its work in a phase, it sends a "done" message to the host. When all done messages have been received, a "go" message is broadcast. This synchronization causes some load imbalance, but since it occurs at such a coarse grain (it occurs only twice per problem solution), it introduces very little performance degradation.

In addition, synchronization points were added within the dc and transient solutions to prevent any PE from getting more than one WR iteration ahead of the rest. While these synchronizations are not functionally necessary, they simplify the bookkeeping needed for waveform references and reduce waveform-storage requirements. This synchronization occurs at a finer grain than that mentioned above and does account for some loss of parallel efficiency (see the beginning of Section 5).

- *Parallel execution improvements*

In an attempt to increase parallelism and improve performance for circuits with significant sequential behavior, a bounded-chaotic (BC) algorithm that favors GS relaxation was implemented. The relaxation is bounded, in that no subcircuit is permitted to be analyzed using input waveforms that are more than one iteration behind the current iteration.

Of the subcircuits assigned to a PE, those that meet the GS ordering requirements are solved first. Whenever no subcircuits on a PE meet the requirements, one of the remaining unsolved subcircuits is selected for analysis using the following criteria:

- Select the subcircuit that would have been solved next if all of the input data were available for this iteration.
- If two or more subcircuits have the same GS level, select from these the subcircuit with the highest percentage of input waveforms available for this iteration.

After one subcircuit is solved out of order, all waiting subcircuits on the PE are again checked to see if any can be solved in order, since new waveforms may have arrived from other nodes in the meantime. This process continues until all subcircuits have been analyzed for the current WR iteration. This approach essentially maintains the GS

ordering, even though the strict data dependencies are compromised, since input waveforms may be from a combination of the current iteration and the previous one. Using the criteria listed above, the BC analysis is likely to be close to GS, and the analysis should still converge reasonably rapidly.

To help guard against false convergence, the inputs of a subcircuit must have been updated by at least one WR iteration before the subcircuit is analyzed again. The implication of this restriction is that an analysis will likely become more "GS-like" as iterations continue. However, because of the small number of WR iterations needed to converge, typically 4 to 15, much performance improvement of BC over GS can be realized. Other alternatives to avoid false convergence include adding one final iteration of true GS relaxation, but this was found to degrade performance significantly. An additional GJ iteration would also work but was not implemented because of the need for a different waveform-storage structure. (Although it has not yet been proved that any of the three methods do as good a job of avoiding false convergence as the conventional algorithm, we have full confidence that a proof is possible.)

With the BC approach, the workload is better balanced for small and irregular circuits, since no PE is idle because of GS serialization, while good use is made of available storage. (As stated earlier, a GJ implementation requires two copies of every waveform and cannot handle large circuits within the memory constraints of V256.) In addition, broken-chain segments can now be analyzed in parallel. GS relaxation guarantees a certain amount of serialization among the PEs when a chain is distributed. Now, those chain segments can be used to better balance the workload across PEs.

The issue of differences in accuracy among the algorithms has been left as an open research topic because of the numerical complexity of the problem. It is simply noted here that the solutions generated by the BC algorithm may not be as accurate as those given by a full GS method. The differences in accuracy are caused by the use of the same convergence criterion by both algorithms. In general, the convergence criterion might not have to be as tight for GS relaxation as for GJ to achieve the same accuracy, since GS relaxation typically approaches the solution more quickly than GJ. Because of the randomness of the BC relaxation, the necessary convergence criterion is not known ahead of time, but it should fall somewhere between that needed for GS and that needed for GJ.

5. Results

This section presents the results of the WR_V256 project. The effect of circuit size on parallel performance using the original GS implementation is summarized in the first subsection. The next subsection expands the set of test cases from the previous subsection to include several circuits from an IBM 16-megabit (Mb) dynamic random access memory (DRAM) design. These circuits represent "real-life" examples, characteristic of current circuit designs. (These experiments also used GS.) The final subsection presents additional results obtained with the BC relaxation improvement described at the end of Section 4.

- *Parallel efficiency*

A suite of ALU circuits, ranging in size from nearly 300 to more than 70 000 FETs, was used to investigate the impact of circuit size on parallel performance. The smallest of these circuits, a four-bit ALU, called ALU_4, served as the fundamental building block for all larger ALU circuits. Of these circuits, only ALU_4 was able to run on a single Victor PE because of memory limitations.

Parallel performance is expressed in terms of PARallel Efficiency, PAR_E, which indicates how closely an application approaches ideal performance. As an application is distributed over many PEs, some of the cycles of every PE are lost because of communication and load imbalance. Parallel efficiency, the ideal run time divided by the actual run time, measures this loss. The ideal parallel run time for each job is defined as the run time of the job on one PE divided by the number of PEs actually used. Parallel efficiency is then

$$PAR\_E = \frac{IDEAL\ RUN\ TIME}{T_p} = \frac{T_s}{n}\frac{1}{T_p}, \quad (3)$$

where T_p is the time to execute on the parallel machine, T_s is the time to execute on a uniprocessor, and n is the number of processors.

Note that ideal run times for the larger simulations cannot be calculated in the straightforward way given above, since the simulations cannot be executed on a single PE. Ideal run times for large circuits were calculated by solving them on a more powerful "PE" (a 3090S processor) and using the 3090S processor run time to estimate the run time on a single Victor PE. By running ALU_4 on both a single Victor PE and a 3090S processor, we determined that a 3090S processor is approximately 17 times faster than a single Victor PE for WR_V256 analyzing the ALU_4 circuit. Ideal run times for Victor were calculated for the other ALU circuits by multiplying the 3090S run times by 17 and then dividing by the number of PEs. Note that this approach assumes that every circuit runs the same mix of instructions, with similar memory reference behavior. This assumption is reasonable for the ALU suite of circuits, since they are all built from the ALU_4 circuit. With this approach, values for PAR_E can be determined and compared.

Figure 4 shows PAR_E, using 64 and 256 Victor PEs, for different circuit sizes. Note that the circuit sizes are

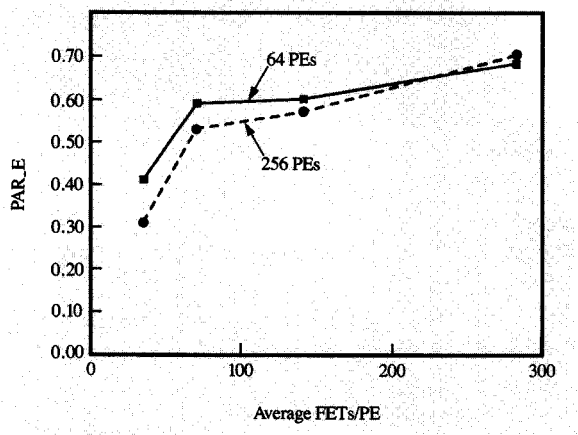

Figure 4

Parallel efficiency (PAR_E) using 64 and 256 Victor PEs for ALU circuits.

Table 1 Test case circuit characteristics and speedup for Gauss–Seidel relaxation on V256.

Circuit name	*Number of FETs*	*Number of nodes*	*Relative speedup*
ALU_4*	282	157	4.76
ALU_8*	564	312	9.18
ALU_16*	1128	622	17.34
ALU_32*	2256	1242	26.35
ALU_64*	4512	2482	48.96
DSRM†	6085	2803	10.71
ALU_128*	9024	4738	79.05
ECC†	16080	5610	38.25
ALU_256*	18048	9250	137.19
QUAD2†	22304	7992	13.43
LBLOCK†	22534	5601	70.38
SPINE†	30639	13526	14.11
ALU_512*	36096	18274	147.73
BLOK2†	46591	11749	188.7
ALU_1024*	72192	39682	181.9
BLOK4†	93182	23352	175.1
CENTX†	186364	46595	126.48

* ALU circuits
† DRAM circuits

displayed in terms of average number of FETs per PE. These figures clearly illustrate that as more and more work is allocated to each PE, the overall parallel efficiency improves. Also, the PAR_E for a lightly loaded system is noticeably lower for 256 PEs than for 64. This implies that the size of the system affects performance for lightly loaded PEs; i.e., performance is lost because of the increased communication in the larger system. For the more heavily loaded cases, there is no discernible difference in PAR_E. For a fully loaded system, the loss of performance is most likely caused by load imbalance and communication overhead, e.g., the cycles needed to pack and unpack messages. Some work remains, in order to quantify the contributions of communication and load imbalance to the loss of parallel efficiency.

- *Performance results*

After the completion of the first set of tests described above, the test suite of circuits was broadened to include several from an IBM 16-Mb DRAM design. These circuits, chosen at random from a 500 000-transistor design, are representative of present-day circuit designs. Data for each circuit are given in **Table 1**, along with speedup for GS relaxation on V256.

It is important to understand how the speedup values were derived. Since none of the DRAM circuits could be run on a single Victor PE, it was not possible to calculate speedup for execution on n processors by the normal rule (run time on one processor divided by run time on n processors). However, all circuits could be run on a 3090S system. Therefore, the execution times of the circuits on the 3090S system were measured and multiplied by a factor of 17 (see the previous subsection) in order to approximate the times that the jobs would have taken on a single Victor PE. This factor is not precise and may vary because the instruction mix and memory reference behavior required for the circuit solutions may vary. Since the behaviors of the 3090S system and the Victor PE are functions of instruction mix and memory access pattern, the relative system performance may differ from circuit to circuit. The speedup presented in Table 1 and Figure 8, discussed later—(run time on 3090S system × 17) ÷ run time on Victor—should be understood in light of the preceding discussion.

The concept of parallel signatures was introduced to help us better understand these results and identify inherent topological limits to parallelism in a circuit. Like the concept of parallel profiles presented in [31], parallel signatures attempt to graphically show how much parallelism is available in an application. Since the definition of parallel signatures has been tailored to address the type of parallelism supported by WR_V256 (i.e., that determined by its data), parallel signatures, as defined below, indicate the *sustainable* amount of parallelism determined by the interconnections among subcircuits, independent of code execution. Parallel profiles are more general in that they show all possible parallelism in a program while it is executing given data, on the basis of data-flow analysis. **Figures 5** and **6** show parallel signatures for the ECC and ALU_1024 circuits, respectively, running on 256 processors. The x axes represent the indices associated with the GS circuit levels. For each level, the

y-axis shows the average number of subcircuits per PE assigned to that level. (The y-axis is dependent on number of PEs because our hope is to eventually extend parallel signatures so that they may predict which type of relaxation to use. Normalizing to PE count gives a more uniform interpretation of the graph, as discussed in the following paragraph.) In deriving parallel signatures, rather than assigning a subcircuit to only one level, e.g., the earliest level in which it could run for GS relaxation, we divide each subcircuit among all of the levels in which it could be run without causing other subcircuits to wait. For example, subcircuit X in Figure 1 is assigned, one-half each, to levels 2 and 3, and subcircuit Y is similarly assigned to levels 1 and 2. That is, a fractional entry is made to every level in which a subcircuit could be run before its outputs are needed. To determine the average number of subcircuits per level per PE, the sum for each level is simply divided by the number of PEs used to analyze the circuit. (For example, from Figure 5 one can see that, on average, approximately 0.1 subcircuit from each of levels 13 to 17 of the ECC circuit was assigned to each PE. Therefore, there are approximately 26 total subcircuits at each of levels 13 to 17 in the ECC circuit.) One drawback of the signature, in this form, is that it contains no information about the relative computational requirements of each subcircuit.

In general, one would expect circuits to run more efficiently when they have parallel signatures that display a number of subcircuits per Victor PE at each level somewhat greater than one. In such cases, it is likely that there will be at least one subcircuit available in each PE to satisfy GS ordering constraints throughout the analysis. (Note that the cutoff value depends on the distribution of subcircuit analysis times. For the ideal case when all times are equal, the number of subcircuits per PE equals one.) Four of the circuits that show relatively poor parallel performance (ECC, QUAD2, SPINE, and DSRM) have parallel signatures that display significant sequential behavior. Of these, only the signature for the ECC circuit is shown. For example, in Figure 5, which shows the parallel signature for the ECC circuit using 256 PEs, the number of subcircuits per PE available for analysis is less than one at all levels. GS serialization forces many of the processors to be idle, which could be a large percentage of the total analysis time. Therefore, it is not surprising that these circuits exhibit poor parallel performance when solved using a GS algorithm. In contrast, Figure 6 shows the parallel signature for the ALU_1024 circuit. The parallel signature for ALU_1024 shows a high degree of parallelism for all of the GS levels.

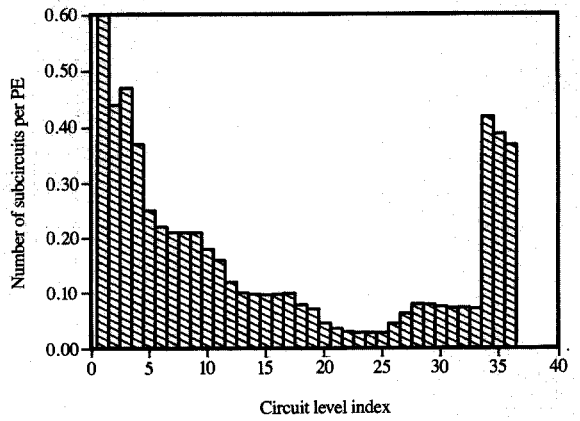

Figure 5

ECC-circuit parallel signature on 256 Victor PEs.

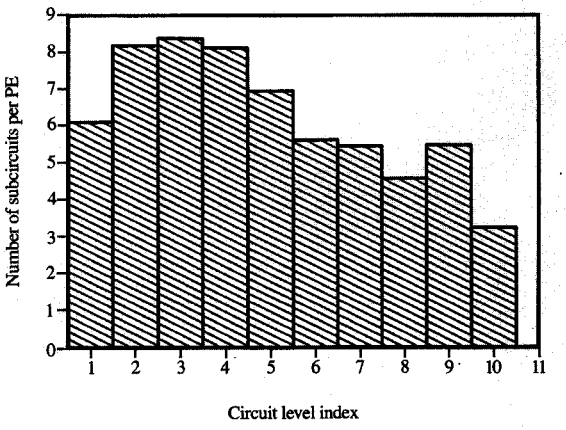

Figure 6

1024-bit ALU parallel signature on 256 Victor PEs.

- *Parallel execution improvements*

Figure 7 compares the performance of WR_V256 using a BC relaxation algorithm with the performance using GS, for five circuits. Application of the BC algorithm improves the parallel performance of WR_V256 on all of these circuits except ALU_1024, which exhibits good parallel execution with a pure GS algorithm and for which the BC algorithm performs essentially the same. The BC algorithm departs from GS only when a processor runs out of work, which rarely happens for the ALU_1024 circuit. The program does incur some minimal additional overhead when running the BC algorithm, but it is insignificant, and

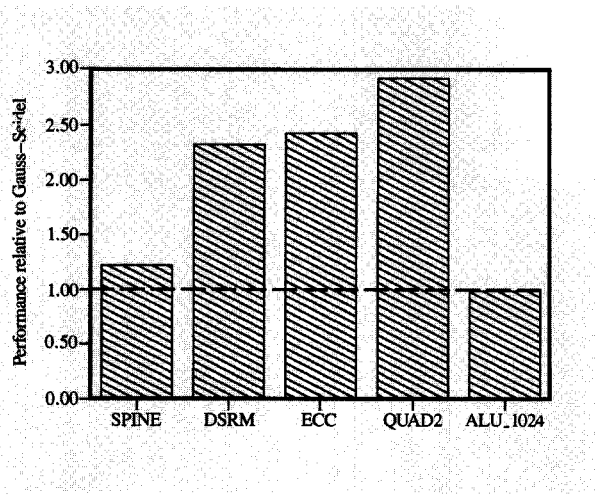

Figure 7

Bounded-chaotic relaxation performance relative to Gauss–Seidel relaxation.

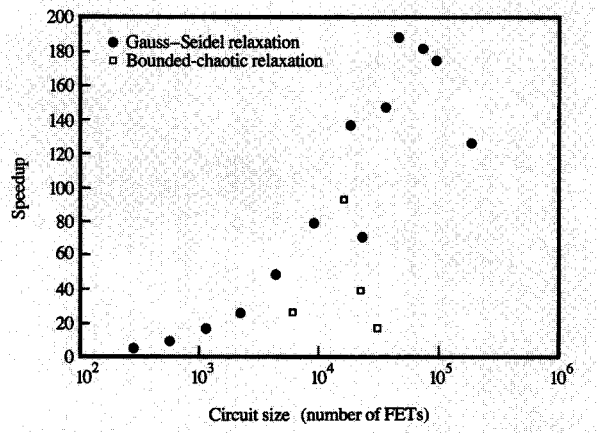

Figure 8

Speedup for all circuits investigated, as a function of circuit size.

independent of circuit size for those circuits maintaining GS relaxation. Circuit QUAD2 yields the most improvement, with a gain of nearly 3×.

Figure 8 shows the speedup of WR_V256 for all circuits investigated. The four circuits with parallel signatures that suggest significant serial behavior have been solved using BC relaxation. The figure shows, in general, that as problem size increases, more parallelism can be exploited, and speedups of 120 to 190 have been achieved for large problems (over 35 000 FETs).

An interesting observation is that the improvements seen with the BC algorithm mirror those presented elsewhere for investigations with the GJ algorithm. In particular, [19] has shown run-time improvements of a "full-window" GJ approach over a similar GS approach in the range of 1× to 3× when there are sufficient PEs to solve one subcircuit per PE. (These improvements are calculated from results given in Table VI of that reference.) Limiting each PE to one subcircuit provides a reasonable upper bound for GJ speed improvements. The maximum speedup is determined in part by the total number of PEs that can be used, the distribution of computational requirements among the subcircuits, the number of GS levels, and the presence of feedback. The same range of improvement is seen for the BC algorithm presented here, even though each PE processes multiple subcircuits.

Our work therefore supports the conclusion that a full implementation of GJ relaxation is not needed to achieve substantial parallel speedup over GS. The BC compromise incurs little additional overhead and offers similar performance results for many circuits. In addition, those circuits with sufficient parallelism can still be executed with the more efficient (numerically) GS relaxation.

6. Concluding remarks

The WR_V256 project has demonstrated that highly parallel machines can be used for circuit simulation, especially for large circuits. For the Victor V256 processor, significant speedup for circuits containing 35 000 to 190 000 transistors was observed. The speedup was possible, even when a more serial relaxation algorithm (GS) was used, because each processing element was given enough work to achieve high parallel efficiency. This work has extended previous research by investigating the effects of a BC relaxation algorithm. The BC algorithm reduces the cost of GS serialization, while retaining much of the GS convergence advantage, and has achieved parallel speedups over GS similar to those obtained elsewhere with a full GJ implementation. The BC implementation requires less memory for waveform storage than GJ, thereby permitting larger circuits to be analyzed. The BC implementation departs from GS relaxation only when a processor would otherwise be idle. Therefore, when the BC algorithm is applied to large circuits with sufficient parallelism, the process may be equivalent to the more rapidly converging GS algorithm.

We hope to broaden this work to better account for the communication and load-balance components of the parallel-efficiency loss, to investigate ways to achieve even higher parallel efficiencies through improved partitioning schemes for circuits with feedback, and to develop better static load-balancing strategies that consider both

interprocessor communication and dynamic switching activity of each subcircuit.

Acknowledgment

We thank all members of the Victor project for their tireless efforts to provide a fully functional, usable, and reliable parallel processor. In particular, we thank Mark Giampapa for the monitor design, Tom Murakami for assistance with the Victor hardware, and Gail Irwin for the file-system software. Without their help, this project would not have been so successful.

References

1. L. W. Nagel, "SPICE2: A Computer Program to Simulate Semiconductor Circuits," *Memo. No. ERL-M520*, University of California, Berkeley, May 1975.
2. W. T. Weeks, A. J. Jimenez, G. W. Mahoney, D. Mehta, H. Qassemzadeh, and T. R. Scott, "Algorithms for ASTAP—A Network Analysis Program," *IEEE Trans. Circuit Theory* **CT-20**, 628–634 (1973).
3. J. White and A. Sangiovanni-Vincentelli, "RELAX2: A New Waveform Relaxation Approach for the Analysis of LSI MOS Circuits," *Proceedings of the IEEE International Symposium on Circuits and Systems*, 1983, pp. 756–759.
4. T. J. LeBlanc, T. J. Cockerill, P. J. Ledak, H. Y. Hsieh, and A. E. Ruehli, "Recent Advances in Waveform Relaxation Based Circuit Simulation," *Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers*, October 1985, pp. 594–596.
5. Y. Fukui, H. Yoshida, and S. Higono, "Supercomputing of Circuit Simulation," *Proceedings of Supercomputing '89*, ACM, Reno, NV, November 1989, pp. 81–85.
6. G. Bischoff and S. Greenberg, "CAYENNE: A Parallel Implementation of the Circuit Simulator SPICE," *Proceedings of the IEEE International Conference on Computer Aided Design*, November 1986, pp. 182–185.
7. H. Yoshida, S. Kumagai, I. Shirakawa, and S. Kodama, "A Parallel Implementation of Large Scale Circuit Simulation," *Proceedings of the IEEE International Symposium on Circuits and Systems*, June 1988, pp. 164–173.
8. A. Hamilton and C. Dyson, "Porting SPICE to the INMOS IMS T800 Transputer," *Technical Note 52*, INMOS Corporation, Bristol, UK, 1988.
9. J. H. Goncalves Romero and R. Weiss, "A New Cellular VLSI Architecture Based on Transputer with Application to Circuit Simulation," *Proceedings of the Fourth European Signal Processing Conference*, September 1988, Grenoble, France, pp. 919–922.
10. C.-P. Yuan, R. Lucas, P. Chan, and R. Dutton, "Parallel Electronic Circuit Simulation on the iPSC® System," *Proceedings of the IEEE 1988 Custom Integrated Circuits Conference*, 1988, pp. 6.5.1–6.5.4.
11. G.-C. Yuan, "PARASPICE: A Parallel Circuit Simulator for Shared-Memory Multiprocessors," *Proceedings of the 27th ACM/IEEE Design Automation Conference*, 1990, pp. 400–405.
12. T. J. Kazmierski and Y. Bouchlaghem, "Hierarchical Solution of Linear Algebraic Equations on Transputer Trees for Circuit Simulation," *Proceedings of the European Conference on Circuit Theory and Design*, IEE, Brighton, UK, September 1989, pp. 42–45.
13. T. Reus, "Acceleration of Circuit Simulation on a Parallel Transputer Workstation," *Proceedings of the Fifteenth EUROMICRO Symposium on Microprocessing and Microprogramming*, September 1989, Cologne, W. Germany, North-Holland, pp. 731–737.
14. S. Mattisson, L. Peterson, A. Skjellum, and C. L. Seitz, "Circuit Simulation on a Hypercube," *Proceedings of the Fourth Conference on Hypercubes, Concurrent Computers, and Applications*, Golden Gate Enterprises, Los Altos, CA, March 1989, pp. 1297–1301.
15. T. J. Kasmierski, *Techniques for Circuit Simulation, Computer Aided Tools for VLSI System Design*, Peter Peregrinus, London, 1987, Ch. 3.
16. A. Ruehli, *Circuit Analysis, Simulation and Design*, North-Holland, Amsterdam, 1987.
17. J. White and A. L. Sangiovanni-Vincentelli, *Relaxation Techniques for the Simulation of VLSI Circuits*, Kluwer Academic Publishers, 1986.
18. C. H. Carlin and A. Vachoux, "MOSART: A Large Scale Time-Domain Simulator Based on Waveform Relaxation," *Proceedings of the European Conference on Circuit Theory and Design (ECCTD 87)*, North-Holland, Paris, September 1987, pp. 227–233.
19. D. W. Smart and T. N. Trick, "Waveform Relaxation on Parallel Processors," *Int. J. Circuit Theory & Appl.* **16**, 447–456 (1988).
20. J. T. Deutsch and A. R. Newton, "MSPLICE: A Multiprocessor-Based Circuit Simulator," *Proceedings of the IEEE International Conference on Parallel Processing*, May 1984, pp. 207–214.
21. J. T. Deutsch and A. R. Newton, "A Multiprocessor Implementation of Relaxation-Based Electrical Circuit Simulation," *Proceedings of the ACM/IEEE 21st Design Automation Conference*, Albuquerque, NM, 1984, pp. 350–357.
22. L. Peterson and S. Mattisson, "Tradeoffs in Partitioning for Waveform Relaxation on Multicomputers," *Proceedings of the IEEE International Symposium on Circuits and Systems*, 1990, pp. 1581–1584.
23. A. Sangiovanni-Vincentelli, *Parallel Processing and Applications*, chapter entitled "Parallel Processing for Simulation of VLSI Circuits," Elsevier Science Publishers B. V. (North-Holland), 1988.
24. R. A. Saleh, K. A. Gallivan, M.-C. Chang, I. N. Hajj, D. Smart, and T. N. Trick, "Parallel Circuit Simulation on Supercomputers," *Proc. IEEE* **77**, 1915–1931 (1989).
25. J. White, R. Saleh, A. Sangiovanni-Vincentelli, and A. R. Newton, "Accelerating Relaxation Algorithms for Circuit Simulation Using Waveform Newton, Iterative Step Size Refinement, and Parallel Techniques," *Proceedings of the IEEE International Conference on Computer-Aided Design*, November 1985, pp. 5–7.
26. D. G. Shea, W. W. Wilcke, R. C. Booth, D. H. Brown, Z. D. Christidis, M. E. Giampapa, G. B. Irwin, T. T. Murakami, V. K. Naik, F. T. Tong, P. R. Varker, and D. J. Zukowski, "The IBM Victor V256 Partitionable Multiprocessor," *IBM J. Res. Develop.* **35**, 573–590 (1991, this issue).
27. *Parallel C Users' Guide*, 3L Ltd., Peel House, Ladywell, Livingstone EH54 6AG, Scotland, February 1988.
28. W. J. Dally and C. L. Seitz, "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks," *IEEE Trans. Computers* **C-36**, 547–553 (1987).
29. J. Yantchev and C. R. Jesshope, "Adaptive, Low Latency, Deadlock-Free Packet Routing for Networks of Processors," *IEE Proc.* **136**, 178–186 (1989).
30. Y. H. Levendel, P. R. Menon, and S. H. Patel, "Special Purpose Computer for Logic Simulation Using Distributed Processing," *Bell Syst. Tech. J.* **61**, 2873–2909 (1982).
31. Arvind, D. E. Culler, and G. K. Maa, "Assessing the Benefits of Fine-Grain Parallelism in Dataflow Programs," *Int. J. Supercomputer Appl.* **2**, 10–36 (1988).

Received October 11, 1990; accepted for publication August 12, 1991

Thomas A. Johnson *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598.* Mr. Johnson received the B.S. degree in physics from Rensselaer Polytechnic Institute in 1968, the M.S. degree in physics from Vassar College in 1974, and the M.S. degree in electrical engineering from the University of Vermont in 1978. He joined IBM at the Product Test Laboratory in Poughkeepsie in 1968, working on several projects relating to system electromagnetic compatibility and noise susceptibility. Since 1971, he has worked in the field of computer-aided circuit analysis and design. In 1978 Mr. Johnson transferred to the IBM East Fishkill facility to work on development of computer-aided electrical package design tools. Currently he is a research staff member at the IBM Thomas J. Watson Research Center.

Deborra J. Zukowski *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598.* Ms. Zukowski is a member of the Modular Microsystems group at the IBM Thomas J. Watson Research Center. After receiving her S.B. degree in electrical engineering from MIT in 1983, she worked at DEC's Eastern Research Laboratory as an internal consultant for performance issues of serial machines. In 1985, she joined the IBM Research Division, where she continued work with uniprocessor performance techniques. Ms. Zukowski has been investigating parallel circuit simulation since January 1989. Her interests include design and implementation of highly parallel architectures and applications, and performance issues of both parallel and serial processors.