Acceleration of Relaxation-Based Circuit Simulation Using A Multiprocessor System

PATRICK ODENT, LUC J. M. CLAESEN, MEMBER, IEEE, AND HUGO DE MAN, FELLOW, IEEE

Abstract—**This paper presents several new methods for the electrical-level simulation of digital VLSI MOS circuits on a shared-memory multiprocessor system. A new parallel algorithm, the overlapped phases algorithm, for the efficient simulation of circuits containing feedback loops, is presented. The algorithm is based on data flow scheduling and local relaxation of the feedback loops. A new method for the partitioning of large pass-transistor networks is discussed. The method is based on the signal flow direction in the elements. This partitioning allows an efficient simulation of these large networks on a multiprocessor system. Parallel element evaluation and the time segment pipelining method, two methods to increase the performance of the parallel circuit simulator, are explained. Simulation tests with actual circuits show a substantial acceleration for the new methods.**

I. Introduction

ACCURATE circuit simulation remains very important in the design process of integrated circuits. Conventional simulators (SPICE [12], ASTAP [22]) are not suited for the simulation of VLSI circuits containing several thousands of transistors. New circuit simulators (RELAX [10], SWAN [4], TOGGLE [9], SPLICE [18]) based on relaxation techniques have been developed in recent years, giving a speedup of more than one order of magnitude.

However, simulation times can still be very large and delay the circuit design. One of the ways to reduce the execution time further is the implementation of the circuit simulators on multiprocessor systems. Relaxation-based techniques decompose the large circuits into a collection of small blocks. This is interesting for parallel processing since each block can be simulated on a different processor. Implementations of relaxation-based simulators on multiprocessor systems are reported by several research groups [4], [6], [11], [13], [17], [19], [23], [25], [28]. The programs are based on a scheduling mechanism to distribute the simulation of subcircuits on different processors. This method works very well for large circuits which tend to be quite "wide."

Multiprocessor circuit simulators are intended for very large circuits, of the order of 100 000 transistors. What is important for the simulation on parallel computers is the structure of these large circuits [3]. They are most likely less uniform than typical smaller circuits and may contain several "difficult" circuits like buses, feedback loops, memory circuits, registers, large blocks of pass-transistor logic, and analog parts which can lead to large subcircuits. These circuits are hard to solve and need special techniques for an efficient simulation on a multiprocessor. So far, this problem has not received much attention in the literature.

Several improvements have been proposed to increase the efficiency of the parallel relaxation algorithms. It is possible to combine a parallel relaxation algorithm with a parallel version of the direct method [14], [26], [28]. Pipelining of computed results between different tasks has been used as a method to realize temporal parallelism [25]. In the time point pipelining method [25] transfer of results is done for each time point. In the parallel time point method [19], [24] a large part of the computation of each time point is done concurrently. These methods give only a moderate improvement in simulation speed, because the additional parallelism is limited and the overhead associated with the scheduling and synchronization is significant.

The original contributions presented in this paper are a number of new algorithms for the efficient simulation of digital VLSI MOS circuits on a shared-memory multiprocessor system. They are implemented in the circuit simulator CSWAN and have been tested on a number of actual circuits. The results are compared to the overlapped WR iterations (OWRI's) algorithm [4], [13]. This algorithm is based on the data flow principle and exploits as much parallelism as possible at the level of the subnetwork simulations. This algorithm is used as a basis of comparison for the other algorithms. The new algorithms are the following.

- The overlapped phases algorithm is an extention of the OWRI's algorithm. It allows an efficient simulation of circuits which contain feedback loops on a multiprocessor system.
- A new partitioning algorithm for large subcircuits is based on the signal flow direction in the elements. Due to this partitioning, several parts of one large subnetwork can be simulated concurrently.
- An algorithm in which the parallelism during the element evaluation is combined in a dynamic way with the parallelism of the relaxation method. This dy-

Manuscript received October 12, 1989. This work was supported by the EC under Grant ESPRIT-1058. This paper was recommended by Associate Editor A. E. Ruehli.

P. Odent and L. J. M. Claesen are with the VSDM Division, IMEC, B-3030 Leuven, Belgium.

H. De Man is with the VSDM Division, IMEC and the Katholieke Universiteit Leuven, B-3030 Leuven, Belgium.

IEEE Log Number 9036677.

namic combination of both parallel methods improves the runtime.
- The time segment pipelining algorithm which provides an efficient way to increase the simulation speed without the tremendeous overhead of other pipelining methods.

First of all, we give a description of the parallel hardware which is used for our research work. In Section III we introduce the waveform relaxation method. In Section IV, the OWRI's method is explained. The overlapped phases algorithm for circuits containing feedback loops is presented in Section V. The new partitioning method for large subnetworks is given in Section VI. The two methods which exploit small grained parallelism are discussed in Section VII. Finally, some conclusions are given.

II. Hardware

The system that has been used for our research work is the Sequent Balance 8000 [20]. Since the hardware strongly influences implementation aspects of a parallel program, we give an introduction on the Sequent Balance 8000. The system is a general purpose parallel computer. Ten 32-b microprocessors have access to the shared memory pool of 8 Mbyte through one 32-b common system bus. Each processor is supported by a floating point unit, memory management unit, and 8 Kbyte cache memory to limit the bus contention. The way to initiate parallel programs is to start several child processes from one parent process. Each child process works on the data in a shared memory. Locks are used to prevent that two processes write in the same memory location at the same time. All interprocess communication is done through the shared memory. There are no explicit message-passing functions.

III. The Waveform Relaxation Method

The program CSWAN is based on the waveform relaxation (WR) method [10]. This section gives a brief description of this method. The electrical simulation problem to verify the behavior of integrated circuits is formulated as a large system of nonlinear ordinary differential equations (ODE's). The ODE system is of the form

$$\dot{q}(v(t), u(t)) - f(v(t), u(t)) = 0, \qquad v(0) = V_0 \tag{1}$$

with q the sum of charges at each node, f the sum of currents charging the capacitance at each node, u the input voltages, and v the unknown node voltages.

In the WR method the system of coupled nonlinear differential equations is transformed into a sequence of nonlinear differential equations in one unknown. In the Gauss–Seidel WR method the following equations are solved for v_i^k, with k the WR iteration number for the entire time interval of window

$$\forall i \; \dot{q}_i(v_1^k, \cdots, v_i^k, v_{i+1}^{k-1}, \cdots, v_n^{k-1}, u) - f_i(v_1^k, \cdots, v_i^k, v_{i+1}^{k-1}, \cdots, v_n^{k-1}, u) = 0. \tag{2}$$

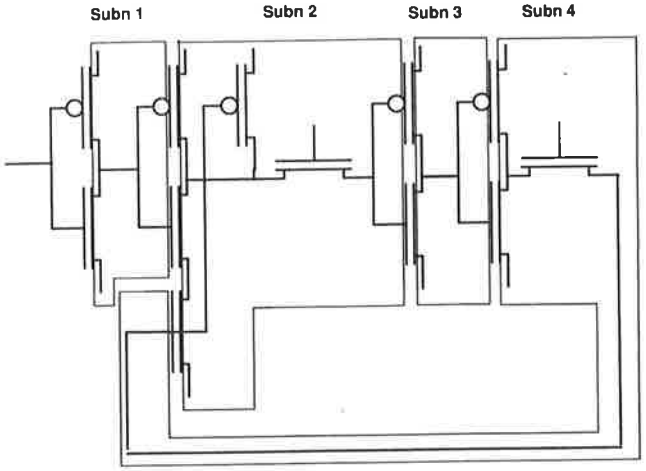

Fig. 1. Network partitioning.

These equations are solved iteratively until convergence.

Slow convergence can occur with this point-wise partitioning [10], due to strong coupling between some variables. In the blockwise partitioning several tightly coupled variables are grouped together. The partitioning of the differential equations is equivalent to the partitioning of the circuit in subcircuits. Each subcircuit is then simulated with a standard circuit simulator, based on the direct method [12]. To take into account the coupling between the subcircuits, WR iterations have to be performed until convergence.

The following network elements are considered in CSWAN:

- MOSFET transistors;
- resistors and capacitors;
- junction diodes;
- grounded voltage sources.

The network partitioning in CSWAN is based on dc unilateral subnetworks [4].

Definition 1: A dc unilateral subnetwork S is a connected group of nodes in the graph which is constructed as follows. Create a vertex for each internal node of the network and an edge between two vertices A and B if they form a drain/source connection of some transistor or if they are the terminals of some resistor.

The elements of a subnetwork are the transistors for which a drain or source is a node of the subnetwork and the other elements for which a terminal is a node of the subnetwork. The subnetwork partitioning is illustrated in the circuit of Fig. 1, which contains four subnetworks. Some other definitions are the following.

Definition 2: A node is an input node of a subnetwork if it is not a node of the subnetwork and if it is a connection for a transistor or resistor of the subnetwork.

Definition 3: A transistor is a fan-out transistor of subnetwork S if a node of subnetwork S is a gate connection for the transistor and if the transistor is not an element of the subnetwork S.

Definition 4: A fan-out node of a subnetwork is either a source, drain, or bulk connection for a fan-out transistor of the subnetwork or a terminal node of a floating capacitor of the subnetwork; and the node must not be a node or input of the subnetwork.

IV. The Overlapped WR Iterations (OWRI's) Algorithm

The parallelism in the WR method is obvious. The process to simulate one subnetwork, in one WR iteration is a single task that can be assigned to a processor and several of these tasks can be executed concurrently. There is, however, a problem of how to distribute the tasks over the different processors. We have investigated a number of algorithms for the scheduling of tasks over the processors. The first algorithm, the leveled subnetworks algorithm [16], is a straightforward parallelization of the sequential version. It contains global synchronization points between each level of subnetworks and each WR iteration. This synchronization reduces the overall performance. These bottlenecks are eliminated in the overlapped WR iterations (OWRI's) algorithm. It is based on the data flow principle and allows the simultaneous simulation of subnetworks in different levels and WR iterations. This algorithm is discussed in this section. It forms the basis of comparison for the other algorithms in this paper.

The data flow principle means that a task is executed as soon as possible, i.e., from the moment that all the data which are needed to execute the task are available. The data flow principle allows to exploit automatically the parallelism in the application: when several tasks become executable at the same time, they can be distributed among the available processors.

In the case of the circuit simulator, a task is the simulation of one subnetwork in one WR iteration. The data needed to execute the task are the waveforms of the input and fan-out nodes of the subnetwork. A subnetwork (the task) can be simulated (executed) as soon as all the waveforms (the data needed for the task) are available. The conditions for a subnetwork S to be simulated in WR iteration k are the following.

1) The waveforms of the input nodes of the subnetwork S must be computed in the same WR iteration k.
2) The waveforms of the fan-out nodes of the subnetwork S must be computed in the previous WR iteration $k - 1$.

The conditions for a task to be executable can be represented in a data flow graph or task dependency graph, see Fig. 2. Each vertex in the graph represents a task: the simulation of a subnetwork in a given WR iteration. The edges represent the conditions for a task to be executable.

The algorithm is not a master–slave configuration in which one master process searches simulatable subnetworks, and gives them to slave processes to simulate them, as proposed in [5]. The algorithm is based on a distributed scheduler. Each processor executes the same scheduling algorithm, which is shown in the following:

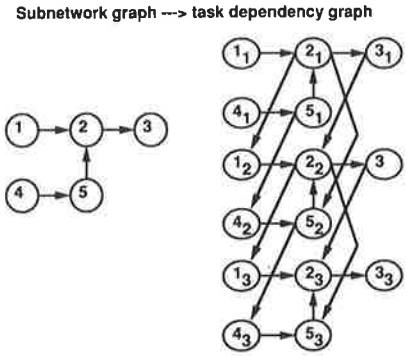

Fig. 2. The task dependency graph of the OWRI's algorithm.

```
Par_process() {
  while ( not converged ) {
    subn= take_from_stack() ;
    if ( subn ) {
      simulate( subn );
      schedule_new_subn( subn );
    }
  }
}.
```

There is one shared stack. All the subnetworks that can be simulated are placed on this stack. Each process can take a subnetwork from this stack to simulate it. After the simulation of the subnetwork, new simulated subnetworks are searched for. This is done by verifiing the output tasks of the executed task in the task dependency graph to see whether all the waveforms which are needed are available. If such subnetworks are found, they are placed on the shared stack.

A problem in this algorithm is the convergence detection since several WR iterations are simulated at the same time. The idea to solve the problem is to start with the minimum number of WR iterations and to start a new WR iteration from the moment a subnetwork is found that did not converge in the highest WR iteration [27]. Notice that this is in fact a violation of the data flow principle. A new WR iteration can be started from the moment there are enough waveforms available. This is not done until there is a real need to do it. A new WR iteration is not started until a subnetwork is detected that does not converge in the last WR iteration that is required at that moment during the simulation. This way of scheduling jobs is in fact "demand-driven" instead of data flow driven.

The OWRI's algorithm has been tested on the Sequent Balance 8000. The measured elapsed "wall-clock" time during the simulation of several circuits is given in Table I. The real speedup S_N is the ratio of the sequential run time T_{seq} and the runtime T_N of the parallel algorithm executed on a given number of processors N. The real speedup for the OWRI's algorithm is also given in Table I. The overhead of the parallel implementation is small

TABLE I
SIMULATION TIME AND REAL SPEEDUP FACTOR OF THE OWRI'S ALGORITHM

Circuit	#MOS	T_{seq} (sec)	T_1 (sec)	T_2 (sec)	T_4 (sec)	T_6 (sec)	T_8 (sec)	S_1	S_2	S_4	S_6	S_8
ALU2	114	684	685	360	218	210	210	1.0	1.9	3.1	3.3	3.3
ALU8.	456	3387	3398	1720	905	656	533	0.99	2.0	3.7	5.2	6.4
CSA.	404	2941	2945	1477	750	534	440	1.0	2.0	3.9	5.5	6.6
BOOTH	1539	4481	4514	2271	1145	796	609	0.99	2.0	3.9	5.6	7.4

because the runtime of the parallel version executed on one processor is only slightly longer than the sequential runtime. The acceleration is very dependent on the size and configuration of the circuit: on eight processors it ranges from a factor three to more than seven.

One might remark that there is still a global synchronization point for all the processes between the simulation of different windows. There are two reasons why this synchronization is not removed. The first reason is that the simulation of the circuit in the different windows is highly sequential. The circuit must be simulated up to convergence in the current window before the simulation in the next window can be started. The second reason is the memory use. Windowing is also intended to reduce the memory use. Simulating several windows in parallel does not preserve this property.

V. The Overlapped Phases Method

Feedback loops form a hard problem in relaxation-based circuit simulators. If no special care is taken for the feedbacks, the number of WR iterations to simulate the circuit can become very large, making the simulator inefficient [4]. Several methods to handle feedback loops have been proposed [3], [4]. Most methods are based on the local relaxation of the subnetworks in the feedback loop. In this way global iterations due to iterations needed to simulate a feedback loop to convergence are avoided.

The number of subnetworks in a feedback loop can be very different. It is even possible that for some circuits, such as finite state machines, nearly the whole circuit is in one large feedback loop. To be efficient, the parallel version of the simulator should be able to exploit the parallelism within the simulation of one feedback loop. The simulation of one feedback loop has to be divided in several subtasks. A new algorithm based on data flow scheduling to solve this problem is explained in this section. First we will explain the method for a uniprocessor. Then the data flow algorithm to exploit the parallelism will be explained. The section concludes with some experimental results.

5.1. Sequential Version

The first step in the simulation of a circuit which contains feedback loops is the search of strongly connected components (SCC) in the subnetwork graph.

Definition 5: SCC: Two subnetworks are placed in the same SCC if there exists a directed path in the subnetwork graph from subnetwork A to subnetwork B and from subnetwork B to subnetwork A.

The algorithm to find the SCC's is a depth first search algorithm [21] which is of linear complexity. After the search of the subnetworks and the SCC's, the SCC's are leveled so that a SCC A is placed before SCC B if SCC A has nodes which are inputs of subnetworks in SCC B.

The sequential WR algorithm for the simulation of circuits with feedback loops is shown in the following:

```
/ * FW-phase */
for ( all SCC's of circuit )
  if ( one subn in SCC )
    simulate( subn );
  else
    while ( SCC not converged)
      for ( all subn of SCC )
        simulate(subn);
/* WR-phase */
  while ( circuit not converged )
    for ( all SCC's of circuit )
      for ( all subn of SCC )
        simulate(subn).
```

The algorithm consists of two major parts:

- the fundamental waveform (FW) phase;
- the WR phase.

In the FW phase "functional correct" waveforms are computed. This is done by local relaxation of the feedback loops. After leveling and ordering the SCC's from input to output, they are simulated. If a SCC contains only one subnetwork, the subnetwork is simulated. If a SCC contains several subnetworks, they are simulated iteratively until convergence. To simulate the subnetworks of the SCC in the correct order the feedback loops are opened and the subnetworks are ordered in the preprocessing phase.

In the WR phase the capacitive coupling between subnetworks is taken into account. The subnetwork simulations are iterated until convergence of the circuit, i.e., until the waveforms of all nodes converge.

5.2. Parallel Version

The parallelism in the previous algorithm is obvious. In the FW phase SCC's can be simulated in parallel and in the WR phase subnetworks can be simulated in parallel. This is, however, not sufficient since a SCC can contain several subnetworks. It is even possible that nearly the whole circuit is placed in one very large SCC. To be efficient the parallel version should be able to simulate multiple subnetworks of one SCC in parallel. It is also possible to start the WR phase before all the fundamental waveforms have been computed. When the fundamental waveforms of two levels of SCC's have been computed, the WR phase can be started. This is a "pipelining" of the FW phase and the WR phase. For this reason we call the algorithm the overlapped phases algorithm.

The algorithm is an extention of the OWRI's algorithm which uses data flow scheduling. Three different jobs have to be considered: SCC's, subnetworks in the FW phase and subnetworks in the WR phase. The simulation of a subnetwork in the FW phase and in the WR phase are considered to be two different tasks because the conditions for the tasks to become executable are different. Also, the scheduling that has to be performed after the execution of the task is different.

A task is executable if all the inputs which are needed to execute the task are available. For the three tasks the conditions to be executable have to be set up. They are a direct result of the different algorithms in the simulation process. The conditions for the three tasks are the following.

1) The simulation of a SCC in the FW phase:
   - a SCC can be simulated if all its SCC inputs which are not external inputs are computed in the FW phase.
2) The simulation of a subnetwork in the FW phase:
   - the SCC, of which the subnetwork is part of, must be able to be simulated (see the previous point);
   - the subnetwork inputs must be computed in the current local relaxation iteration, except if the input node is a feedback input or an external input.
3) The simulation of a subnetwork in the WR phase:
   - the subnetwork inputs have to be computed in the current WR iteration, except if they are external inputs or feedbacks;
   - the fan-out nodes and feedback inputs have to be computed in the previous WR iteration;
   - two additional conditions are needed for the "pipelining" of the FW phase and the WR phase. First of all, the SCC, of which the subnetwork is part of, must be simulated to convergence in the FW phase;
   - for the first WR iteration in the WR phase, the other condition is that the fan-out nodes and feedback input nodes of the subnetwork must be computed in the FW phase.

Given these tasks and the conditions for the tasks to be executed, a task dependency graph can be constructed. This is shown on Fig. 3 for the small circuit of Fig. 1 with four subnetworks and one feedback loop. Assume that the SCC needs three local iterations, which is a typical number to reach convergence, in the FW phase and the total circuit needs two additional WR iterations in the WR phase to reach convergence. The data flow graph is constructed by creating a vertex for each task in the simulation process. These are the simulation of the two SCC's in the FW phase and the simulation of the four subnetworks in two WR iterations in the WR phase. The simulation of SCC 2 in the FW phase can be expanded in the simulation of three times the three subnetworks of the SCC. The edges between the tasks represent the data flow conditions for the tasks to be simulated.

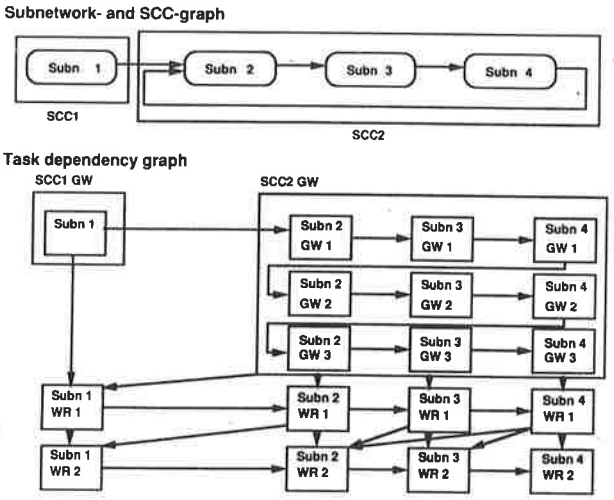

Fig. 3. The tasks dependency graph of the overlapped phases algorithm.

Based on this relationship between the tasks, a data flow based scheduler can be used to execute all tasks in the correct order on a parallel computer system. The simplified parallel scheduler which is executed concurrently on all the processors is shown in the following:

```
Par_process( ) {
  while ( not converged ) {
    /* Take SCC from stack */
    while( SCC= take_SCC_from_stack( ) )
      for( all subn's of SCC )
        if ( subn_simul( subn ) )
          place_on_stack( subn );
    /* Take subn in FW-phase */
    while ( subn= take_FW_subn_from_stack( ) ) {
      Simulate( subn );
      Schedule_new_tasks(subn);
      }
    /* Take subn in WR-phase */
    while ( subn= take_WR_subn_from_stack( ) ) {
      Simulate( subn );
      Schedule_new_subn(subn);
    }
  }
}.
```

There are three stacks with executable tasks. Each process can take tasks from each of the stacks. After the execution of the task, some scheduling has to be performed. All the fan-out tasks in the task dependency graph have to be verified. If such a task is executed it is placed on the appropriate stack.

5.3. *Experimental Results*

This overlapped phases algorithm has been tested on the Sequent Balance 8000. Several circuits containing feedback loops have been simulated on a number of processors ranging from 1 to 8. The measured simulation times T_N and real speedup factors S_N, of which N is the number of processors, are given in Table II.

TABLE II
SIMULATION TIMES AND REAL SPEEDUP FACTORS OF THE OVERLAPPED PHASES ALGORITHM

Circuit	# MOS	T_{seq} (sec)	T_1 (sec)	T_2 (sec)	T_4 (sec)	T_6 (sec)	T_8 (sec)	S_1	S_2	S_4	S_6	S_8
OSC	19	83	83	83	84	84	85	1.0	1.0	0.99	0.99	0.98
SIPO	160	3256	3260	2522	2293	2254	2252	1.0	1.3	1.4	1.4	1.4
CCH2	114	958	960	508	317	295	293	1.0	1.9	3.0	3.3	3.3
CCH8	456	4973	4983	2526	1313	918	716	1.0	2.0	3.8	5.4	6.9
ADDACC	528	2115	2157	1071	549	396	417	1.0	2.0	3.9	5.3	5.1

By comparing the results of the parallel algorithm executed on one processor, with the sequential algorithm, we can conclude that the parallel implementation has only a very small scheduling overhead. The speedup of the parallel overlapped phases algorithm for the simulation of circuits containing feedback loops, strongly depends on the size and configuration of the circuits. On eight processors, the measured speedup ranges from one for a small circuit to more than six for the large circuits. There is no speedup for the ring oscillator (OSC). This is because a node of the last subnetwork is a feedback input of the first subnetwork. This means that with the Gauss–Seidel relaxation method, the first subnetwork of the oscillator can only be simulated in a higher WR iteration when the last subnetwork has been simulated in the current WR iteration. So, an overlapped simulation of the WR iterations is not possible. For the same reason it is not possible to exploit parallelism in the local relaxation process. The ADDACC circuit, a 4-b adder, and accumulator placed in a global feedback loop, shows a much better parallelism. This is because several subnetworks which are placed in the global feedback loop, are simulated in parallel with the overlapped phases algorithm. If the local relaxation process of this global feedback loop would not be executed in parallel, the speedup would be much smaller.

The same algorithm can be used for the feedback loop free circuits. For circuits without feedback loops, the overlapped phases algorithm is the same as the OWRI's algorithm.

VI. LARGE SUBNETWORKS

As explained above, the WR method partitions the circuit in a number of subcircuits. Several methods for the blockwise partitioning of the circuit in subcircuits have been developed:

user partitioning [10];
partitioning based on dc-unilateral networks [4];
diagonal dominance Norton partitioning [25].

For special circuits such as shifters, RAM's, and pass-transistor logic, most methods give rise to large subcircuits. These large subcircuits are not suited for parallel processing. If a large subcircuit is simulated by only one processor, the efficiency of the parallel execution can become very bad.

The new method proposed in this paper partitions the large subnetworks in smaller blocks. This is suited for parallel processing since all the blocks can be simulated concurrently on different processors. Moreover, the simulation time can further decrease because each block is simulated with its own optimal timestep and with less overhead to solve the linear equations in the subcircuit simulator.

The large subnetworks are partitioned into smaller blocks making use of signal flow information [1], [8]. For each subnetwork a signal flow direction can be defined from the input nodes (the gate terminals of the MOSFET's of the subnetwork) to the output nodes of the subnetwork.

The following rules are used to determine the signal flow direction in the transistors of a subnetwork (see Fig. 4).

1) If the drain or source of a transistor is a subnetwork output, the data flow direction is from the other terminal to the output node.
2) If the drain or source of a transistor is a subnetwork input, the data flow direction is from the input node to the other terminal.
3) If multiple elements are connected to one node, all except one element are directed in the same direction and one element is not yet directed, then, that undirected element will have the opposite direction.
4) If all transistors connected to a node y are set with equal direction except for two unset transistors $T1$ and $T2$, and the gates of $T1$ and $T2$ are complements of each other, and the node y is not an input or an output, then both transistors $T1$ and $T2$ must transmit signals in the opposite direction at node y with respect to the set transistors.

It is not always possible to direct all transistors. So, the direction of some transistors will be undefined or bidirectional. A preprocessing of the circuit with the DIALOG expert system [1] could be used to direct more transistors making use of a more extended set of rules. However, this limited set of rules allows us to show the properties of the method.

Once the signal flow direction of the transistors is known, the subgroups are searched for. The definition of a subgroup is based on the signal flow direction of the elements. There is a subgroup for each output of the subnetwork.

Definition 6: A subgroup is a group of nodes in the direct signal flow graph of a subnetwork. It contains all the nodes of the subnetwork which can, based on the signal flow, influence a given output node of the subnetwork.

Subgroups are searched starting from the outputs of the subnetwork. The way to search all those nodes is based on a depth first search algorithm in the opposite direction of the signal flow, starting from an output node. The output node and all the subnetwork nodes that can be reached

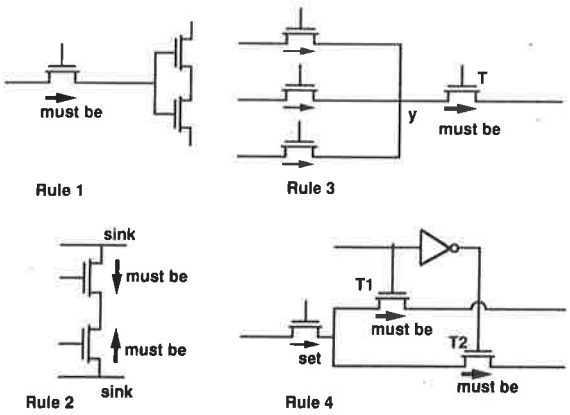

Fig. 4. The signal flow direction.

by traversing the directed graph through set transistors in the direction opposite to the signal flow or through unset transistors, are placed in the same subgroup. Note that this partitioning in subgroups is an overlapped partitioning, i.e., a node can be placed in more than one subgroup. If a node can influence two or more outputs, it will be placed in the subgroup of each output.

In the shifter of Fig. 5, the signal flow of the transistors and the partitioning in subgroups is shown. The inverters at the outputs of the shifter are different subnetworks and only two out of eight subgroups of the shifter are represented. There is a small overlapping of the two subgroups.

It is not difficult to extend the parallel scheduler of the overlapped phases method with the method to simulate large subnetworks. The additional part of the scheduler has the following form:

```
/* Take a subnetwork to simulate */
. . .
if ( large subnetwork )
  place_subgroups_on_stack( subn );
. . .
/* Take a subgroup to simulate. */
while ( subgr= take_subgroup_from_stack() ) {
  Simulate( subgroup );
  if ( All subgroups of the subn simulated ) {
    schedule_new_tasks( sub );
  }
}.
```

When a large subnetwork is taken from the stack to be simulated, its subgroups, which are searched in the preprocessing, are placed on the stack of subgroups. All processes can take a subgroup from that stack to simulate it. After the simulation of the subgroup, a shared locked counter of the subnetwork, of which the subgroup is a part, is incremented by one. If the number of simulated subgroups equals the number of subgroups in the subnetworks, the same functions are executed as if the subnetwork was simulated without partitioning in subgroups.

The results in Table III are the speedup factors of two circuits with large subnetworks. As can be seen from these

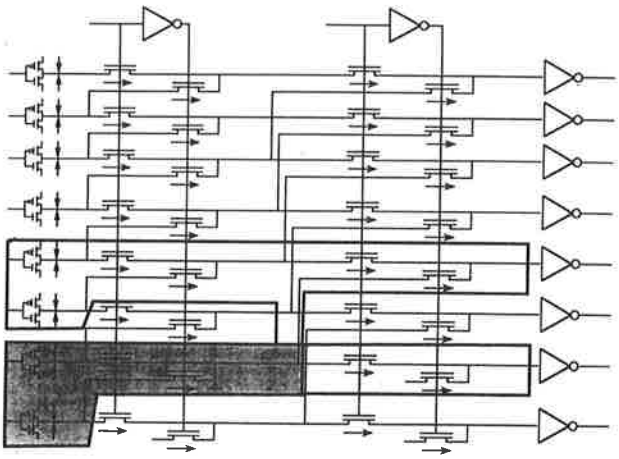

Fig. 5. Signal flow and subgroups in a shifter.

TABLE III
SIMULATION TIMES AND REAL SPEEDUP FACTORS FOR CIRCUITS CONTAINING LARGE SUBNETWORKS

Circuit	# MOS	T_{seq} (sec)	T_1 (sec)	T_2 (sec)	T_4 (sec)	T_6 (sec)	T_8 (sec)	S_1	S_2	S_4	S_6	S_8
SHIFTER	240	1453	1458	762	413	348	256	1	1.9	3.5	4.2	5.7
MCALU	292	3237	3268	1656	954	843	779	1	2.0	3.4	3.9	4.2

results, a significant speedup can be obtained by executing this algorithm on a parallel computer. The simulation of the shifter on eight processors is more than five times faster than on one processor.

VII. Small Grained Parallelism

Up to now only the parallelism at the level of subcircuits (SCC, subnetwork, subgroup) is exploited. This is very efficient, but the acceleration largely depends on the size and configuration of the circuit. It is possible to increase the speed of the simulation process by parallelizing several other parts of the program. Two new methods have been developed in our research work.

1) In the first method, the parallelism at the level of subcircuits is combined dynamically with the parallelism in the subcircuit simulator. When there are not enough subcircuits to keep all the processors busy, the parallel direct method is used to increase the parallelism.
2) The second method, called the time segment pipelining method, makes an additional speedup possible by dividing the simulation interval into smaller segments. The simulations of the subcircuits in these smaller time segments can be pipelined. This gives a significant speed improvement.

These two methods are further explained in this section.

7.1. Parallel Element Evaluation

The subnetworks are simulated with the direct method. Several parallel versions of the direct method have been

proposed. However, only a few results of the combination of the parallel WR method and the parallel direct method are published. In [26] a method with dedicated hardware for the element evaluation is proposed. In [28] both the model evaluation phase and LU-decomposition are parallelized. The processors are statically partitioned in groups. All the processors of one group work together to simulate one subcircuit.

We propose a method in which the processors are not statically partitioned in groups. However, processors can be grouped dynamically during the simulation to perform parallel model evaluation when there are not enough simulatable subcircuits to keep all the processors busy with only the parallel WR method. All the tasks, subnetwork simulations and element evaluations, are dynamically distributed over all the processors.

Since in a relaxation method the circuit is partitioned into smaller subcircuits, the major portion of the computation in the subcircuit simulator is the element evaluation. Experimental results [16] indicate that 60–70% of the total simulation time is taken by element evaluations. For each timestep, all the elements can be evaluated in parallel. Since for several elements the stamps of the MNA matrix can be in the same entries, some synchronization is needed. The speedup will be limited since only one part of the direct method is done in parallel. Theoretical computations [16], based on Amdahl's Law, indicate that if the parallelism in the WR method at the level of the subnetwork simulation is not exploited, an acceleration of 2.2–2.7 can be reached with the parallel element evaluation.

The parallel element evaluation is implemented with a shared stack for each process. On this stack we place all the elements that have to be evaluated for the current timestep of a subnetwork under simulation on a given processor. If another processor becomes idle, i.e., there is no subnetwork for that processor, it can help another processor with the element evaluation. The idle processor can look on the stack of elements of other processors. If such an element is found, it is evaluated and its stamp is added in the MNA matrix. It is obvious that several steps of the process have to be protected with lock operations. Due to these lock operations the acceleration is smaller than the theoretical value given above. The synchronization overhead to get a new element is large compared to the time to evaluate the element. Several techniques of the parallel direct method could be used to improve the implementation. For example, merging several element evaluations into one larger task to limit the synchronization overhead [7].

As long as there are enough subnetworks to distribute over the processors, there is no parallel element evaluation. Only when there are idle processors does the parallel evaluation of the elements start on some processors. This dynamic combination of the parallel WR method and the parallel direct method is efficient since it gives an optimal tradeoff between parallelism and efficiency. For large circuits, nearly 100% of time is spend in the parallel WR mode, while for the small circuits, up to 100% of time can be spent in the parallel direct method mode.

Table IV shows some simulation times of the new method. The third column gives the sequential execution time. In the fourth column are the results of the OWRI's algorithm on eight processors. The fifth column gives the simulation times if only the parallelism in the element evaluation is exploited. This is 1.6 times faster than the sequential version. The last column gives the dynamic combination of the parallel WR method and the parallel element evaluation. We see that for small circuits or for circuits with little parallelism that the method can reduce the simulation time with a 30% comparison to the OWRI's method. For large circuits the additional gain with parallel element evaluation is limited. This is because in these circuits there are a lot of subnetworks which can be simulated in parallel. So, the processors are only idle during very small periods in which they can do element evaluations for other processors. If more than eight processors can be used, a significant acceleration is also expected for these large circuits. However, experiments on a larger system are needed to verify if bus or shared stack contention are not becoming a bottleneck.

7.2. *Time Segment Pipelining*

A method to increase the parallelism of the WR method is time point pipelining [25]. The results show good speedup factors, but if the simulation times are compared with the simulation times of the method without pipelining, it is clear that the time point pipelining method suffers from scheduling overhead. The simulation time on one processor is 50% larger for the time point pipelining method.

The time segment pipelining (TSPL) method shows much less overhead [4]. In this paper we present an improved version of the method and additional results which indicate that there is an optimal size for the time segment. To explain the method we use the chain of inverters of Fig. 6. In the Gauss–Seidel relaxation method, and taking into account the directionality of the circuit, subnetwork 2 can only be simulated if subnetwork 1 has been simulated. This is because the internal node of subnetwork 1 is an input of subnetwork 2. Also, subnetwork 3 can only be simulated after the simulation of subnetwork 2. This is represented in the first processor usage plot of Fig. 6. This condition is in fact too restrictive. When the first subnetwork is simulated over a certain time segment, the simulation of the second subnetwork can be started in that time segment. At the same time the first subnetwork can be simulated in the next time segment. This is illustrated in the second plot of Fig. 6.

In the TSPL method the whole simulation interval or window is divided in a number of time segments. The simulation of subnetworks is done over only one time segment. A subnetwork is now able to be simulated if all inputs are computed in the same time segment for this waveform iteration and if all fan-out nodes are computed in the same time segment for the previous waveform iteration. An additional condition is that the subnetwork

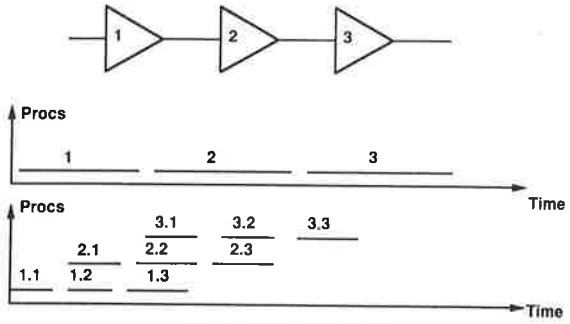

Fig. 6. Three invertors.

TABLE IV
SIMULATION TIMES WITHOUT AND WITH PARALLEL ELEMENT EVALUATION

Circuit	# MOS	T_{seq} (sec)	T_{sWR} (sec)	T_{sDir} (sec)	$T_{sWR-Dir}$ (sec)
ALU2	114	684	210	431	152
ALU8	456	3387	533	2146	521
CSA	404	2941	440	1733	413
OSC	19	83	85	52	52
ADDACC	528	2115	417	1320	405
SHIFTER	240	1458	258	909	242
MCALU	292	3268	779	488	643

has to be simulated in the previous time segment. Feedback inputs of the subnetwork are considered to be fan-out nodes.

For the TSPL method there is an optimal length for the time segment. It can be seen from Table V, which shows the execution times for several simulations with time segments from 5 to 100 ns, that the simulation time is dependent on the size of the time segment. However, the size is not very critical. Only for a too small size of the time segment are the scheduling and communication cost becoming so large that the increase in parallelism is not large enough to improve the speedup. This tradeoff between parallelism and overhead is similar to the tradeoff between convergence speed and overhead in the choice of the window size.

For an optimal length of the time segments, from 10 to 20 ns, the improvement in simulation time is more than 30% for several circuits. For large circuits, which are already efficiently simulated with the OWRI algorithm, there are no idle processors to exploit the additional parallelism. However, on a larger system with more processors, an equivalent acceleration could be observed for these large circuits, provided that the bus and shared stack contention do not become bottlenecks.

Fig. 7 shows the processor usage plots for the simulation of the ALU2 circuit with the OWRI's algorithm and the TSPL method. The average processor usage is much higher for the pipelining method and the reduction in simulation time is significant.

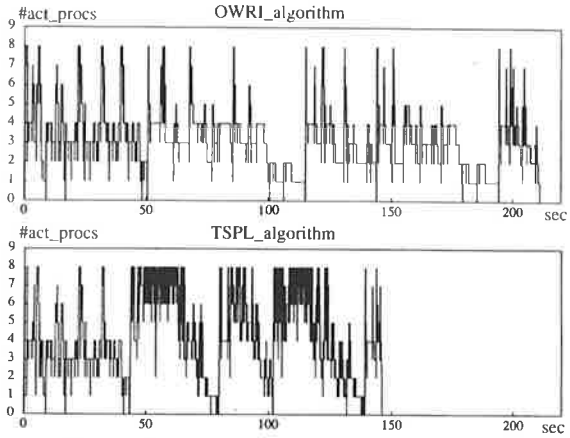

Fig. 7. The processor usage of the TSPL method.

TABLE V
SIMULATION TIMES OF THE TSPL METHOD

	# MOS	2ns (sec)	5ns (sec)	10ns (sec)	25ns (sec)	OWRI (sec)
ALU2	114	229	145	151	200	210
ALU8	456	3077	710	498	479	533
CSA	404	4554	565	390	395	440

VIII. CONCLUSIONS

In this paper several new algorithms for the efficient electrical level simulation of large VLSI circuits on multiprocessor systems are presented. An efficient parallel scheduler for the simulation of circuits with feedback loops is explained. A new static partitioning method for large pass-transistor networks is proposed. Two techniques to exploit smaller grained parallelism are presented: a dynamic combination of the parallel WR method and parallel element evaluation and the TSPL method. The program CSWAN exploits parallelism at several levels: SCC's, subnetworks, subgroups, time segments, and element evaluations. Experimental results show the power of the new algorithms.

REFERENCES

[1] I. Bolsens, W. De Rammelaere, L. Claesen, and H. De Man, "Electrical debugging of synchronous MOS VLSI circuits exploiting analysis of the intended logic behaviour," in *Proc. 26th ACM/IEEE Design Automation Conf.*, June 1989, pp. 513–518.

[2] L. O. Chua and P. M. Lin, *Computer Aided Analysis of Electronic Circuits: Algorithms and Computational Techniques.* Englewood Cliffs, NJ: Prentice-Hall, 1975.

[3] P. Debefve, F. Odeh, and A. E. Ruehli, "Waveform techniques," in *Circuit Analysis, Simulation and Design*, A. E. Ruehli, ed. Amsterdam, The Netherlands: North-Holland, 1987, part 2, pp. 41–127.

[4] D. Dumlugol, P. Odent, J. Cockx, and H. De Man, "Switch-electrical segmented waveform relaxation for digital MOS VLSI and its acceleration on parallel computers," *IEEE Trans. Computer-Aided Design*, vol. CAD-6, pp. 992–1005, Nov. 1987.

[5] D. Dumlugol, "The segmented waveform relaxation method for mixed-mode simulation of digital MOS VLSI circuits," Ph.D. dissertation, Katholieke Univ. Leuven, Oct. 1986.

[6] G. K. Jacob, A. R. Newton, and D. O. Pederson, "An empirical analysis of the performance of a multiprocessor-based circuit simu-

lator," in *23rd ACM/IEEE Design Automation Conf. Proc.*, June 1986, pp. 588–593.
[7] G. Jacob, A. Newton, and D. Pederson, "Direct method circuit simulation using multiprocessors," in *Proc. Int. Symp. on Circuits and Systems*, vol. 1, May 1986, pp. 170–173.
[8] N. P. Jouppi, "Derivation of signal flow direction in MOS VLSI," *IEEE Trans. Computer-Aided Design*, vol. CAD-6, pp. 480–490, May 1987.
[9] H. Y. Hsieh, A. Ruehli, and P. Ledak, "Progress on toggle: A waveform relaxation VLSI-MOSFET CAD program," in *Proc. Int. Symp. on Circuits and Systems*, 1985, pp. 213–216.
[10] E. Lelarasmee, A. Ruehli, and A. L. Sangiovanni-Vincentelli, "The waveform relaxation method for the time-domain analysis of large scale integrated circuits," *IEEE Trans. Computer-Aided Design*, vol. CAD-1, pp. 131–145, Aug. 1982.
[11] S. Mattison, "CONCISE: A concurrent circuit simulation program," PhD dissertation, Dep. Appl. Electron., Lund Institute of Technology, Lund, 1986.
[12] L. N. Nagel, "SPICE2: A computer program to simulate semiconductor circuits," Memo. no. ERL-M520, Univ. California, Berkeley, Elec. Res. Lab., May 1975.
[13] P. Odent, D. Dumlugol, and H. De Man, "Hardware acceleration of circuit simulation on a multi-microprocessor system," in *Hardware Accelerators for Electrical CAD*, by T. Ambler, P. Agrawal and W. Moore, eds. Oxford, U.K.: Adam Hilger, 1987, pp. 154–163.
[14] P. Odent, L. Claesen, and H. De Man, "New parallel techniques for simulating MOS circuits with waveform relaxation algorithms in CSWAN," in *Proc. Int. Symp. on Circuits and Systems*, May 1989, pp. 1166–1169.
[15] ——"Feedbackloops and large subcircuits in the multiprocessor implementation of a relaxation based circuit simulator," in *Proc. 26th ACM/IEEE Digital Automation Conf.* 1989, June 1989, pp. 25–30.
[16] P. Odent, "Electrical-level simulation of VLSI MOS circuits using multi-processor systems," Ph.D. dissertation, Katholieke Univ. Leuven, Jan. 1990.
[17] L. Peterson and S. Matisson, "Circuit partitioning and iteration scheme for waveform relaxation on multicomputers," in *Proc. Int. Symp. on Circuits and Systems*, May 1989, pp. 570–573.
[18] R. A. Saleh, J. E. Kleckner, and A. R. Newton, "Iterated timing analysis and SPLICE1," in *Dig. Tech. Papers Int. Conf. Computer-Aided Design*, Sept. 1983, pp. 139–140.
[19] R. A. Saleh, D. Webber, E. Xia, and A. Sangiovanni-Vincentelli, "Parallel waveform-Newton algorithms for circuit simulation," in *Proc. Int. Conf. Computer Design*, 1987, pp. 660–663.
[20] A. Osterhaug, "Guide to parallel programming," Sequent Computer Systems, Inc.
[21] R. Tarjan, "Depth-first search and linear graph algorithms, *SIAM J. Comput.*, vol. 1, no. 2, June 1972.
[22] W. T. Weeks, A. J. Jimenez, G. W. Mahoney, D. Mehta, H. Qua-semzadeh, and T. R. Scott," Algorithms for ASTAP—A network analysis program," *IEEE Trans. Circuit Theory*, vol. CT-20, pp. 628–634, Nov. 1973.
[23] D. Webber and A. Sangiovanni-Vincentelli, "Circuit simulation on the connection machine," in *Proc. 24th ACM/IEEE Design Automation Conf.*, 1987.
[24] 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," in *Dig. Tech. Papers Int. Conf. on Computer-Aided Design*, Nov. 1985, pp. 438–441.
[25] J. White and A. L. Sangiovanni, "Partitioning algorithms and parallel implementations of waveform relaxation algorithms for circuit simulation," in *Proc. Int. Symp. on Circuits and Systems*, June 1985, pp. 229–231.
[26] J. White and N. Weiner, "Parallelizing circuit simulation- A combined algorithmic and specialised hardware approach," in *Proc. Int. Conf. on Computer Design*, Oct. 1986, pp. 438–441.
[27] J. White and A. Sangiovanni-Vincentelli, *Relaxation Techniques for Simulation of VLSI Circuits*, Norwell, MA Kluwer Academic, 1986.
[28] H. Yoshida, S. Kumagai, and I. Shirakawa, "A parallel implementation of large-scale circuit simulation," in *Hardware Accelerators for Electrical CAD*, T. Ambler, P. Agrawal and W. Moore, eds. Oxford, U.K.: Adam Hilger, 1987, pp. 164–173.

*

Patrick Odent received the electrical engineering degree and the Ph.D. degree in applied sciences from the Katholieke Universiteit Leuven, Belgium, in 1985 and 1990, respectively.

In 1985 he joined the VSDM Division of the Interuniversity Microelectronics Center (IMEC), Leuven, Belgium, as a Research Assistant. His research interests are computer-aided design, circuit simulation, and parallel processing.

*

Luc J. M. Claesen (S'77–M'85) received the electrical engineering degree and Ph.D. degree from the Katholieke Universiteit Leuven, Belgium, in 1979 and 1984, respectively.

After graduation in 1979 he joined the ESAT-Laboratory, Katholieke Universiteit Leuven, as a Research Assistant, where he worked in the field of computer-aided design of integrated systems for digital and analog signal processing. In 1984 he joined the IMEC Laboratory in Heverlee, Belgium, where he is heading research in the Design Methodologies Division. Since 1989 he has been an Associate Professor at the Katholieke Universiteit Leuven. His research interests are in computer-aided design, especially formal design and verification methods, as well as their application to dedicated chip architectures for image synthesis systems.

Dr. Claesen is a member of IFIP working group 10.2: "System Description and Design Tools." He received a Best Paper Award at the ICCD 1986 Conference.

*

Hugo De Man, for a photograph and biography please see page 937 of the September 1990 issue of this TRANSACTIONS.