A combined Waveform Relaxation - Waveform Relaxation Newton algorithm for efficient parallel circuit simulation

P. Odent, L. Claesen, H. De Man
IMEC, Interuniversity Micro Electronics Center, VSDM division, Kapeldreef 75, 3030 Leuven, Belgium.

Abstract
This paper presents two new techniques for accelerating circuit simulation. The first technique is an improvement of the parallel Waveform Relaxation Newton (WRN) method. The computations of all the timepoints are executed concurrently. Static task partitioning is shown to be an efficient method to limit the scheduling overhead. The second technique combines in a dynamic way the efficiency of the parallel version of the Waveform Relaxation (WR) method and the parallelism of the new developed parallel WRN algorithm.

1 Introduction
Accurate electrical-level circuit simulation remains very important in the design process of integrated circuits. Conventional circuit simulators, such as SPICE [5] and ASTAP [9], are computationally expensive when applied to large systems. A number of relaxation based circuit simulators, such as RELAX [3], TOGGLE [2], SWAN [1], SPLAX [10], have been developed to reduce the simulation time.

Further speed improvement can be obtained using parallel processors. Relaxation based techniques are particularly well suited for parallel processing since they decompose the large circuits into a collection of small subcircuits [12], [4], [1], [8].

In this paper two new parallel algorithms are introduced: an improved parallel Waveform Relaxation Newton algorithm based on static partitioning of the time points and an optimal combination of the parallel WR method and the parallel WRN method.

This paper is organized as follows. Section two gives a description of the multiprocessor. Section three outlines two relaxation based circuit simulation techniques: the Waveform Relaxation method and the Waveform Relaxation Newton method. Section four discusses the properties of the parallel versions of these methods. An improved parallel Waveform Relaxation Newton method is presented in section five. A new algorithm based on the combination of the Waveform Relaxation method and the Waveform Relaxation Newton method is introduced in section six. The paper concludes with experimental results and conclusions.

2 Hardware
The multiprocessor system used for this research is the Sequent Balance 8000 [6]. The system is a general purpose parallel computer. Ten 32-bit microprocessors have access to the shared memory pool of 8 Mbytes through one 32-bit 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 a parent process. Each child process works on the data in the shared memory. Locks are used to prevent that two processes write in the same memory location at the same time. All inter-process communication is done through the shared memory. There are no explicit message-passing functions.

3 Relaxation techniques
The electrical-level simulation of integrated circuits is formulated as the numerical solution of a large system of nonlinear ordinary differential equations (ODE). In this paper we will consider the following general formulation of these ODE’s [13]:

\[ C(v(t), u(t))v'(t) = f(v(t), u(t)) \quad v(0) = V_0 \]  

where \( v(t) \in R^n \) on \( t \in [0, T] \) is the vector of unknown node voltages; \( u(t) \in R^r \) on \( t \in [0, T] \) is the vector of input voltages; \( C(v(t), u(t)) \) \( \in R^{n \times n} \) is the capacitance matrix; and \( f(v(t), u(t)) \in R^r \) is the vector of branch currents.

Several relaxation based methods to solve this system of ODE’s have been developed. The first method discussed in this paper is the Waveform Relaxation (WR) method [3]. The WR method is an iterative method based on a relaxation scheme which applies decomposition directly to the system of differential equations. The Gauss-Seidel WR method can be described as follows:

- Step 1: Initialization of the relaxation process.
  Set \( k = 0 \) and make an initial guess of the waveforms: \( v_i'(t) \) for \( t \in [0, T] \) so that \( v_i'(0) = v_i \).
- Step 2: Analysis of the decomposed system at the \( k \)-th iteration.
For each \( i = 1, 2, \ldots, n \) solve for \( v_i(t); t \in [0, T] \) from following equation:

\[
\begin{align*}
\sum_{j=1}^{n} C_{ij}(v_{i1}^k, \ldots, v_{ij}^k, v_{ij+1}^{k-1}, \ldots, v_{in}^{k-1}, u)v_j^k + \\
\sum_{j=i+1}^{n} C_{ij}(v_{i1}^k, \ldots, v_{ij}^k, v_{ij+1}^{k-1}, \ldots, v_{in}^{k-1}, u)v_j^{k-1} = \\
f_i(v_{i1}^k, \ldots, v_{ij}^k, v_{ij+1}^{k-1}, \ldots, v_{in}^{k-1}, u)
\end{align*}
\]

with \( u^k(0) = u_0 \).

\( \bullet \) Step 3: Iteration.

Stop the relaxation process if the difference between \( u^k(t) \) and \( u^{k-1}(t) \) is small enough over the simulation interval \([0, T]\), i.e. \( \max_{t \in [0, T]}|u^k(t) - u^{k-1}(t)| \leq \varepsilon \). Otherwise, set \( k = k + 1 \) and go to step 2.

Note that equation 2 has only one unknown variable \( v_i \). The variables \( v_{i1}^{k-1}, \ldots, v_{in}^{k-1} \) are known from the previous iteration and the variables \( v_{i1}^k, \ldots, v_{in}^k \) are computed in the current iteration.

In the WR method each differential equation is solved exactly. However, it is also possible to solve these nonlinear differential equations approximately. If the accuracy of the approximation is obtained by performing one step of the Waveform Newton method, then the convergence of the following nonlinear differential equation

\[
J_f(x)(x^k(t) - x^{k-1}(t)) = -F(x^k(t))
\]

where \( J_f(x) \) is the Frechet derivative of \( F(x) \) with respect to \( x \). Note that \( J_f(x) \) is a matrix-valued function on \([0, T]\).

In the Waveform Relaxation Newton (WRN) method [11], [10] the WR iteration equations are solved approximately by performing one step of the Newton method. The WRN iteration equations can easily be found by applying one step of the Newton method to the WR iteration equations:

\[
\begin{align*}
\sum_{j=1}^{n} C_{ij}(v_{i1}^{k-1}, \ldots, v_{ij}^{k-1}, v_{ij+1}^{k-2}, \ldots, v_{in}^{k-2}, u)v_j^{k-1} + \\
\sum_{j=i+1}^{n} C_{ij}(v_{i1}^{k-1}, \ldots, v_{ij}^{k-1}, v_{ij+1}^{k-2}, \ldots, v_{in}^{k-2}, u)v_j^{k-2} - \\
f_i(v_{i1}^{k-1}, \ldots, v_{ij}^{k-1}, v_{ij+1}^{k-2}, \ldots, v_{in}^{k-2}, u)
\end{align*}
\]

These equations have only one unknown variable \( v_i \). Note that the nonlinear differential equations of the WRN algorithm have been replaced by a time-varying linear differential equation.

\section*{4 Evaluation of parallel WR and WRN}

In recent years parallel computer systems have been used to improve the speed of circuit simulation. Both the WR and the WRN methods have been implemented on multiprocessors. This section evaluates some of these parallel techniques.

The WR method has been implemented on multiprocessor systems by several research groups [4], [3], [7]. 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". However, it has been observed that the speedup depends strongly on the functional parallelism of the circuit.

A parallel program, PSPLAX, based on the WRN is described in [10]. Subcircuit evaluations are distributed over different processors. This method gives a reasonable speedup if the circuit is large and "wide" enough. A technique to increase the parallelism is the Parallel Time Points (PTP) method [10]. In the PTP method several time points are computed in parallel. In the WRN method, the number of time points and their location are known before the simulation of the subnetwork. As can be seen in equation 5, the devices are linearized with voltages of the previous iteration. Therefore the capacitance matrices and some parts of the RHS vectors can be computed in parallel for all time points. Also the LU decomposition can be executed in parallel for all time points. A remaining task must be executed sequentially across timepoints. This remaining task consists of loading the rest of the RHS and the forward and backward substitution.

The simulation of each subcircuit is done with the PTP method, but only a limited amount of time points are computed in parallel. The scheduling of the parallel tasks is based on a shared queue. Therefore the capacitance matrices and some parts of the RHS vectors can be computed in parallel for all time points. The parallel version executed on one processors is 50% slower than the sequential version. This scheduling overhead limits the speedup. On eight processors the speedup factor is only three.

We can conclude by saying that previous research has shown that the parallel relaxation methods are very efficient for large and "wide" circuits, but for smaller circuits there is not enough parallelism to keep all the processors busy. The PTP method can give a lot of additional parallelism, but excessive scheduling overhead limits the speedup. Therefore we propose two new algorithms: an efficient parallel WRN algorithm, based on static task partitioning, and an optimal combination of the parallel WR and the parallel WRN method.

\section*{5 An efficient parallel WRN algorithm}

In this section we present a new parallel WRN algorithm in which only the parallelism resulting from parallel time point computation is exploited. Parallelism at the level of the subnetwork simulations is not used. This allows us to concentrate on the properties of the PTP method. Next section presents an algorithm that combines the parallelism at the two levels.
To limit the scheduling overhead we do not use a shared stack with executable tasks. Instead, the time points of one subcircuit simulation are partitioned statically over a given number of processors. This reduces scheduling cost since each process knows which tasks it has to execute and there is no contention of a shared stack of executable tasks. Load balancing is not a problem since the amount of work does not change very much between different time points in the simulation of the same subcircuit.

The parallel algorithm of the subcircuit simulator, based on a one step Newton method, is shown on figure 1. First the new time points are looked for and placed in a shared array. Then a number of child processes are started which execute the function \texttt{Steps\_compute()} for each process. Each process has an unique identification number \texttt{MyNum}, from 0 up to the number of processes minus one. Based on this number each process knows which time points it has to compute: all the time points of the shared array with index \texttt{MyNum + i * nr\_procs} for \texttt{i = 0...n}. If the time point is beyond the end of the simulation interval the process stops. All processors can start with the computation of the matrix stamps and LU-decomposition for their time point. Then they have to execute the critical section. This critical section can only be executed if the previous time point is completely computed. This synchronization is implemented with a while loop, and a shared array of Boolean variables, called \texttt{tptcomputed[]} for all the fields of the array are initialized on \texttt{FALSE}. If the computation of the \texttt{i}-th time point is finished, the variable \texttt{tptcomputed[i]} is set on \texttt{TRUE}. At this moment the process that computes the next time point will leave the while loop and start executing the critical section. Remark that we do not need to protect this shared variable with a lock since only one process can change the variable and only one process reads the variable until it changes.

\begin{verbatim}
Simulate\_subcircuit( subn) {
    New\_timepoints();
    m_set\_procs( nrproc );
    m_fork( Steps\_compute );
    m_sync();
}

Steps\_compute() {
    MyNum= m_get\_myid();
    for ( tmp= MyNum; timepoint[tmp] <= stop;
        tmp= tmp+nr\_procs ) {
        Fill\_matrix( timepoint[tmp] )
        LU\_decomposition();
        while( tptcomputed[tmp-1] == FALSE )
            /*
            Rest\_of\_RHS(); /* CRITICAL SECTION */
            LU\_RHS(); /* CRITICAL SECTION */
            Solve(); /* CRITICAL SECTION */
            tptcomputed[tmp]= TRUE;
    }
}
\end{verbatim}

Figure 1: Parallel WN algorithm with static data partitioning.

Figure 2 shows the synchronization between four processes during the simulation of one subcircuit. All processors start with the matrix fill in and LU-decomposition of four different time points. Process one, computing the first time point, can immediately continue with the critical section. The other processes have to wait until the voltages of the previous time point have been computed. When process one has computed the voltages of the first time point, process two starts the critical computation of the voltages of the second time point. This synchronization is represented by an arc in the figure. At the same time process one takes a new time point, the fifth, and begins with the matrix fill in and LU-decomposition. Somewhat later, when process two has computed the voltages at the second time point, process three can execute the critical section and process two starts the computation of the next time point. This continues until all the time points have been simulated. From this figure follows immediately that the critical section limits the maximum speedup that can be obtained with this method. From experimental results follows that the critical section is about 10% of the total simulation time.

Figure 2: Processor activity and synchronization during the execution of the parallel WN algorithm on four processors.

Figure 3 shows an experimental processor usage plot, obtained during the simulation of the FBB circuit with the PTP method on eight processors. A processor usage plot shows the number of active processes in function of the elapsed "wall-clock" time during the simulation. The plot shows the simula-

\begin{verbatim}
1
2
3
4
5
6
7
8
Matrix-fill and LU-decomposition
Critical section
Idle time

Figure 3: Processor usage plot of the PTP algorithm on eight processors.
\end{verbatim}
tion of one subcircuit. At the beginning, the processor usage is equal to the number of processors because all the processes can start with the non-critical section of a different time point. After nearly one second, the synchronization between the processes decreases the number of active processes. In steps of 0.1 sec, the length of the critical section, the processor usage increases again to eight. At the end of the simulation the processor usage drops stepwise to zero. This is because the computation of each time point ends with the critical section.

6 The optimal combination

As discussed in section 4, the WR method parallelized at the level of subnetwork simulations is very efficient for large and "wide" circuits. In previous section, it has been shown that the WRN method can be parallelized at the level of the time point computation. However, speedup and efficiency are limited due to the synchronization. Therefore, we present in this section an optimal combination of both methods. The new method gives priority to the parallel WR method since it is the most efficient method and the parallelism is only limited by the "width" of the circuit. But if there are idle processors, the parallelism is increased by simulating some subnetworks with the parallel WRN method.

The combined WP-WRN algorithm is shown on figure 4. The function Par_scheduler() is executed on all the processors of the system. As long as there is no convergence, each process tries to take a subnetwork from the shared stack. There are three possible situations:

1. The process gets a subnetwork and there are no idle processes. In this case the subnetwork is simulated with the direct method by the subcircuit simulator. This is in fact the normal WR method.

2. The process gets a subnetwork from the stack and there are idle processes. Some of these idle processes are chosen to help the process with the simulation of the subnetwork. The subnetwork is simulated with one step of the Waveform Newton method and the PTP method is used to exploit the parallelism.

3. It is also possible that the process does not get a subnetwork from the shared stack. In that case, the process will check if it has been assigned to help another process.

After the simulation of the subnetwork, new subnetworks that can be simulated are searched and placed on the shared stack. This search is executed in the Fork_new_tasks() and is based on the data flow principle. For more information about this part of the program we refer to [7].

This combined WR-WRN algorithm will always take the most efficient amount of parallelism. If the circuit is very wide and there is enough parallelism at the level of subnetworks, the parallel WR method will be used. If the circuit is smaller and there are not enough parallel tasks to use all the processors, then the algorithm will automatically start the simulation of some subnetworks with the parallel WRN method.

7 Experimental results

This section gives some experimental results from simulations with three algorithms: the WR method, the WRN method and the combined WR-WRN method. For each method test results are given for the sequential version and for the parallel version executed on 1, 2, 4 and 8 processors. The computed speedup factors are the ratios of runtime of the sequential algorithm and the runtime of the parallel algorithm on a given number of processors.

Table 1 gives the run times and speedup of the WR method.

Table 1: Simulation times and speedup factors of the WR method.

<table>
<thead>
<tr>
<th>Circuit</th>
<th># MOS</th>
<th>T_seq</th>
<th>T_1</th>
<th>T_2</th>
<th>T_3</th>
<th>T_4</th>
<th>T_5</th>
</tr>
</thead>
<tbody>
<tr>
<td>FBB</td>
<td>15</td>
<td>47s</td>
<td>48s</td>
<td>44s</td>
<td>43s</td>
<td>43s</td>
<td></td>
</tr>
<tr>
<td>ALU2</td>
<td>114</td>
<td>684s</td>
<td>683s</td>
<td>378s</td>
<td>273s</td>
<td>262s</td>
<td></td>
</tr>
<tr>
<td>ALU4</td>
<td>226</td>
<td>1514s</td>
<td>1517s</td>
<td>811s</td>
<td>527s</td>
<td>413s</td>
<td></td>
</tr>
<tr>
<td>INV64</td>
<td>128</td>
<td>459s</td>
<td>459s</td>
<td>230s</td>
<td>116s</td>
<td>60s</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Simulation times and speedup factors of the WR method.
Table 2 gives the run times and the speedup factors of the WRN method. If we compare the sequential run times \( T_{seq} \) of the WRN method with the sequential run times of the WR method given in table 1, we see that there is no significant difference in simulation time between the WRN method and the WR method. The parallel version of the WRN method exploits only the parallelism at the level of the time point computation, as explained in section five. As can be seen on table 2 the speedup is more or less independent of the simulated circuit but the speedup is rather small. The speedup factor on 8 processors is not more than 3.2.

Table 2: Simulation times and speedup factors of the WRN method.

<table>
<thead>
<tr>
<th>Circuit</th>
<th>( T_{seq} )</th>
<th>( T_1 )</th>
<th>( T_2 )</th>
<th>( T_3 )</th>
<th>( T_4 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>FBB</td>
<td>75s</td>
<td>76s</td>
<td>53s</td>
<td>31s</td>
<td>24s</td>
</tr>
<tr>
<td>ALU2</td>
<td>654s</td>
<td>657s</td>
<td>419s</td>
<td>273s</td>
<td>206s</td>
</tr>
<tr>
<td>ALU4</td>
<td>1358s</td>
<td>1363s</td>
<td>846s</td>
<td>586s</td>
<td>418s</td>
</tr>
<tr>
<td>INV64</td>
<td>524s</td>
<td>526s</td>
<td>320s</td>
<td>215s</td>
<td>166s</td>
</tr>
</tbody>
</table>

\[
\begin{array}{ccccccc}
\text{Circuit} & S_2 & S_1 & S_0 & S_6 \\
\hline
FBB & 1.0 & 1.4 & 2.4 & 3.1 \\
ALU2 & 1.0 & 1.6 & 2.4 & 3.2 \\
ALU4 & 1.0 & 1.6 & 2.3 & 3.2 \\
INV64 & 1.0 & 1.6 & 2.4 & 3.2 \\
\end{array}
\]

Table 3 gives the expected results of the combined WR-WRN method. For small circuits, the simulation times and speedup factors are much better than the WR and WRN methods. Take for example circuit ALU2, the simulation time on eight processors is reduced from 262 sec to 122 sec. This is because processors which are idle in the WR method are used to compute some time points in parallel, based on the WRN method. The overhead introduced in this combined WR-WRN method is very limited. There is nearly no difference in simulation time between the "optimal" sequential version and the parallel version executed on one processor. For circuits which are simulated efficiently with the parallel WR method, the combined WR-WRN method also gives the same performance since in this case the WR method is used, see for example circuit INV64.

Table 3: Simulation times and speedup factors of the combined WR-WRN method.

<table>
<thead>
<tr>
<th>Circuit</th>
<th>( T_{seq} )</th>
<th>( T_1 )</th>
<th>( T_2 )</th>
<th>( T_3 )</th>
<th>( T_4 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>FBB</td>
<td>47s</td>
<td>48s</td>
<td>31s</td>
<td>18s</td>
<td>14s</td>
</tr>
<tr>
<td>ALU2</td>
<td>684s</td>
<td>685s</td>
<td>360s</td>
<td>201s</td>
<td>122s</td>
</tr>
<tr>
<td>ALU4</td>
<td>1514s</td>
<td>1516s</td>
<td>797s</td>
<td>421s</td>
<td>240s</td>
</tr>
<tr>
<td>INV64</td>
<td>459s</td>
<td>461s</td>
<td>231s</td>
<td>118s</td>
<td>61s</td>
</tr>
</tbody>
</table>

8 Conclusions

In this paper we present two new algorithms for parallel circuit simulation. After presenting the mathematical background of some relaxation based simulation techniques, an evaluation of existing parallel implementations is given. Then the improved parallel WRN method is introduced. The method is based on a static partitioning of the time point computations over the available processors. The second algorithm, an optimal combination of the parallel WR method and the improved parallel WRN method, is explained. This combined parallel WR-WRN method provides an optimal combination of the properties of the two methods: the efficiency of the WR method and the additional parallelism of the WRN method. Experimental results show the substantial acceleration of this method.

References