

# WavePipe: Parallel Transient Simulation of Analog and Digital Circuits on Multi-Core Shared-Memory Machines

Wei Dong  
Department of ECE  
Texas A&M University  
College Station, TX 77843  
weidong@ece.tamu.edu

Peng Li  
Department of ECE  
Texas A&M University  
College Station, TX 77843  
pli@neo.tamu.edu

Xiaoji Ye  
Department of ECE  
Texas A&M University  
College Station, TX 77843  
yexiaoji@neo.tamu.edu

## ABSTRACT

While the emergence of multi-core shared-memory machines offers a promising computing solution to ever complex chip design problems, new parallel CAD methodologies must be developed to gain the full benefit of these increasingly parallel computing systems. We present a parallel transient simulation methodology and its multi-threaded implementation for general analog and digital ICs. Our new approach, *Waveform Pipelining* (abbreviated as *WavePipe*), exploits coarse-grained application-level parallelism by simultaneously computing circuit solutions at multiple adjacent time points in a way resembling hardware pipelining. There are two embodiments in *WavePipe*: backward and forward pipelining schemes. While the former creates independent computing tasks that contribute to a larger future time step by moving backwards in time, the latter performs predictive computing along the forward direction of the time axis. Unlike existing relaxation methods, *WavePipe* facilitates parallel circuit simulation without jeopardizing convergence and accuracy. As a coarse-grained parallel approach, *WavePipe* not only requires low parallel programming effort, more importantly, it creates new avenues to fully utilize increasingly parallel hardware by going beyond conventional finer grained parallel device model evaluation and matrix solutions.

## Categories and Subject Descriptors

B.7.2 [Integrated Circuits]: Design Aids—*simulation*

## General Terms

Algorithms, Design, Performance

## Keywords

Transient Simulation, Parallel Computing, Multi-Core

## 1. INTRODUCTION

The wide spread of multi-core microprocessors is making parallel computing mainstream [1–3]. Unlike conventional supercomputers, modern multi-core processors are of low cost and widely accessible to typical circuit designers.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

DAC 2008, June 8–13, 2008, Anaheim, California, USA  
Copyright 2008 ACM 978-1-60558-115-6/08/0006...5.00

With low on-chip communication overhead and high memory bandwidth as well as continuing technology scaling, multi-core or many-core systems are increasingly in a position to provide needed computing power to address many computationally intensive CAD problems. As one of the most critical forms of pre-silicon simulation and verification, SPICE-like transistor-level transient circuit analysis is indispensable to a broad range of designs including memories, custom digital and analog/RF/mixed-signal ICs [4]. The time consuming nature of transient analysis often makes it a significant design bottleneck, necessitating its parallelization.

There exist a number of parallel simulation approaches, majority of which are fine grained in nature. It is possible to parallelize the key steps in an existing simulation algorithm, e.g. device model evaluation and matrix solution. However, the efficiency of parallel matrix solvers can deteriorate fairly quickly as the number of processor cores increases. On supercomputers and computer clusters, waveform relaxation and other nonlinear relaxation methods have been proposed for parallel circuit simulation [5–7]. However, these methods are not widely used for robust general circuit simulation due to limited convergence properties. The efficiency of the domain decomposition approach in [8] is strongly application dependent, leading to limited applicability. Furthermore, the above two approaches require fine-grained parallel programming, hence high implementation and debugging effort.

As a coarse-grained application-level parallel approach, *WavePipe* is proposed with the recognition of two key needs in parallel application development: 1) the need to exploit domain knowledge to achieve good parallel processing efficiency by overcoming the high inter-core/thread communication overhead in fine-grained parallel approaches, and 2) the need to go beyond conventional fine-grained schemes to create a rich enough set of parallelism to fully utilize increasingly parallel computing platforms. *WavePipe* exploits application-level parallelism along the time axis by simultaneously computing the circuit solutions at multiple adjacent time points using a combination of two novel schemes: backward and forward pipelining schemes. Backward pipelining is employed in conjunction with variable step-size multi-step numerical integration methods; by moving backward along the time axis, it creates additional independent computing tasks that contribute to a larger future time step. Forward pipelining, on the other hand, facilitates predictive computing along the forward direction of the time axis.

*WavePipe* complements and goes beyond what can be offered by parallel device model evaluation and matrix solving and provides orthogonal opportunities for parallel com-

puting. As a coarse grained parallel approach, *WavePipe* requires only moderate modification of existing serial simulation codes and hence is easy to implement and debug. Unlike waveform relaxation and other relaxation methods, *WavePipe* maintains the same convergency property of the standard SPICE transient analysis. Furthermore, it speeds up transient simulation without jeopardizing accuracy.

## 2. BACKWARD PIPELINING

Electronic circuits can be described by the following differential equations in time domain

$$f(x(t)) + \frac{d}{dt}q(x(t)) + u(t) = 0, \quad (1)$$

where  $x(t)$  is the vector of nodal voltages and branch currents,  $u(t)$  is the input,  $f(\cdot)$  and  $q(\cdot)$  are nonlinear functions describing static and dynamic nonlinearities. To solve the above nonlinear differential equations numerically, in transient analysis a numerical integration method, such as Backward Euler (BE) or Trapezoidal Rule (TR), is applied to convert (1) to a sequence of nonlinear algebraic equations. The concept of the local truncation error (LTE) is used to control the errors incurred in numerical integration, resulting in the LTE-based time step control [9]. The idea is to limit the time step size such that a pre-defined local truncation error tolerance is satisfied.

In standard transient analysis, time-domain circuit responses are computed sequentially along the time axis such that for the solution of any time point, the known responses of the preceding points provide a well defined initial condition. At the first glance, in both one-step and multi-step integration methods, the predetermined data dependency seemingly makes it impossible to enable parallel computing along the time axis, as illustrated in Fig. 1. However, as will

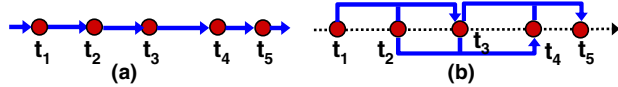


Figure 1: Data dependency in a) one-step, and b) multi-step (2-step) numerical integration.

be described, variable-step size multi-step methods can be indeed exploited for parallel computing, but via some new perspectives.

### 2.1 Variable-Step Size Multi-Step Methods

The multi-step Gear's integration formulae have the following form [10]

$$x_{n+1} = \beta_0 \dot{x}_{n+1} + \sum_{k=1}^p \alpha_k x_{n+1-k}, \quad (2)$$

where  $p$  is the order of numerical integration,  $x_i$ ,  $i = n+1-p, \dots, n+1$  is the circuit response at time point  $i$ ,  $x_{n+1}$  is the unknown circuit response at time point  $(n+1)$ , and  $(\beta_0, \alpha_k)$  are certain coefficients, which are constant in fixed-step Gear's methods. Consider the special case of the two-step

variable time-step Gear's method [11]

$$x_{n+1} = -x_{n-1} \frac{h_{n+1}^2}{h_n(2h_{n+1} + h_n)} + x_n \frac{(h_{n+1} + h_n)^2}{h_n(2h_{n+1} + h_n)} + \dot{x}_{n+1} \frac{h_{n+1}(h_{n+1} + h_n)}{2h_{n+1} + h_n}, \quad (3)$$

where  $h_{n+1} = t_{n+1} - t_n$ ,  $h_n = t_n - t_{n-1}$ . The local truncation error of (3) is

$$\varepsilon_{n+1} = -\frac{h_{n+1}^2(h_{n+1} + h_n)^2}{6 \cdot (2h_{n+1} + h_n)} \cdot x^{(3)}(\tau), \quad (4)$$

where  $\tau$  is in  $[t_n, t_{n+1}]$ , and the third order derivative  $x^{(3)}(\tau)$  may be approximated as  $x^{(3)} \approx 3!DD_3(t_{n+1}, t_n, t_{n-1}, t_{n-2})$ , in which  $DD_3(t_{n+1}, t_n, t_{n-1}, t_{n-2})$  denotes the third order divided difference evaluated at time points  $t_{n+1}, t_n, t_{n-1}, t_{n-2}$ . As a standard Gear2 method, this integration formula is stiffly stable [11].

The above formulae can be extended to a three-step one

$$x_{n+1} = \beta_0 \dot{x}_{n+1} + \alpha_1 x_n + \alpha_2 x_{n-1} + \alpha_3 x_{n-2}. \quad (5)$$

Define a set of new variables:  $T_1 = t_{n+1} - t_n$ ,  $T_2 = t_{n+1} - t_{n-1}$ ,  $T_3 = t_{n+1} - t_{n-2}$ . The coefficients in (5) can be obtained by solving

$$\begin{bmatrix} 1 & 1 & 1 & 0 \\ T_1 & T_2 & T_3 & -1 \\ T_1^2 & T_2^2 & T_3^2 & 0 \\ T_1^3 & T_2^3 & T_3^3 & 0 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \beta_0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad (6)$$

which leads to

$$\begin{aligned} \alpha_1 &= \frac{T_2^2 T_3^2}{(T_2 - T_1)(T_3 - T_1)(T_1 T_2 + T_1 T_3 + T_2 T_3)} \\ \alpha_2 &= \frac{T_1^2 T_3^2}{(T_1 - T_2)(T_3 - T_2)(T_1 T_2 + T_1 T_3 + T_2 T_3)} \\ \alpha_3 &= \frac{T_1^2 T_2^2}{(T_1 - T_3)(T_2 - T_3)(T_1 T_2 + T_1 T_3 + T_2 T_3)} \\ \beta_0 &= \frac{T_1 T_2 T_3}{T_1 T_2 + T_1 T_3 + T_2 T_3} \end{aligned} \quad (7)$$

The local truncation error of (5) is given by

$$\varepsilon_{n+1} = \frac{T_1^2 T_2^2 T_3^2}{24(T_1 T_2 + T_2 T_3 + T_1 T_3)} \cdot x^{(4)}(\tau), \quad (8)$$

where  $\tau$  is in  $[t_n, t_{n+1}]$  and  $x^{(4)}(\tau)$  is approximated by  $x^{(4)} \approx 3!DD_4(t_{n+1}, t_n, t_{n-1}, t_{n-2}, t_{n-3})$ , in which  $DD_4(t_{n+1}, t_n, t_{n-1}, t_{n-2}, t_{n-3})$  denotes the fourth order divided difference evaluated at time points  $t_{n+1}, t_n, t_{n-1}, t_{n-2}, t_{n-3}$ .

In the LTE-based time-step control, the largest time step size that does not exceed a specified local truncation error tolerance is chosen. This strategy ensures that the numerical integration error incurred at each time step is well controlled while the transient simulation can be advanced in time as fast as possible. In the two-step Gear's method presented above, the time step can be estimated based on LTE as follows. For given  $h_n = t_n - t_{n-1}$ ,  $DD_3$  and a specified LTE tolerance  $\varepsilon$ , let the time step to be determined as  $h_{n+1} = kh_n$ ,  $k > 0$ . According to (4), the maximum allowable  $h_{n+1}$  can be determined by solving the following equation

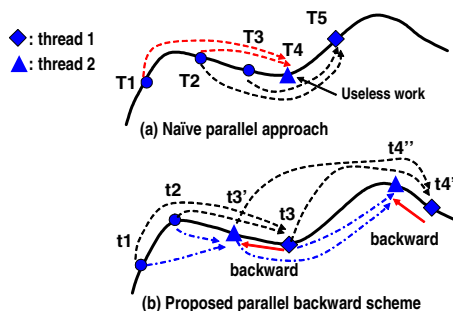
$$\frac{k^2(k+1)^2}{(2k+1)} = \left| \frac{\varepsilon}{DD_3 \cdot h_n^3} \right|. \quad (9)$$

In addition to LTE, other factors may be further considered when choosing a suitable time step. Because nonuniform step-sizes modify the stability region of the integration

method, time step may not be changed too rapidly in order to assure that the stability property does not depart considerably from that of a uniform step-size method. For instance, step-size variations can be constrained such that  $\frac{1}{\alpha} \leq k \leq \alpha$ , where  $\alpha > 0$  and is not too large.

## 2.2 Parallel backward pipelining

Without loss of generality, we present backward pipelining under the context of double-threaded two-step variable time-step integration methods. The discussion can be easily extended to other multi-threaded scenarios with a higher-order integration method. To simplify the discussion, we assume that the third order divided difference  $DD3$  remains constant at different time points although in practice variable  $DD3$  can be easily handled. Under this assumption, the LTE is only a function of the two time steps,  $h_n$  and  $h_{n+1}$ , in (4).



**Figure 2: Parallel double-threaded backward pipelining: a) an naïve approach, and b) the proposed backward pipelining.**

Let us first consider a naïve approach as shown in Fig. 2 (a). The circuit responses at three time points  $T_1 - T_3$  are assumed to be known. Using the solutions at  $T_2$  and  $T_3$  as the initial conditions, a thread may be launched to compute the solution at  $T_5$ . One may attempt to use a second thread to compute the solution at  $T_4$  by using solutions at  $T_1$  and  $T_2$  as the initial conditions. This choice may seemingly make use of two processing elements (threads) to allow for parallel computing. However, a more careful look reveals that the work done by the second thread at  $T_4$  is almost always useless. This is because  $T_5$  is usually beyond  $T_4$  due to the use of the most updated initial conditions. The solution at  $T_4$  only provides an interpolation point between  $T_3$  and  $T_5$  and by itself does not contribute to a faster transient analysis.

To exploit the variable-step size two-step methods (e.g. Gear2) in a more meaningful way, we propose parallel backward pipelining as shown in Fig. 2 (b). While the first thread is solving the transient circuit response at a time point that is determined by the standard numerical integration and LTE step control, a second thread is launched to solve the solution at a preceding time point in parallel, but based on the same latest available initial conditions. To see how this can speed up the transient analysis, let us examine the dependency of the LTE of the two-step numerical integration on the two time steps,  $h_n$  and  $h_{n+1}$ , in (4). The partial derivatives of the LTE with respect to  $h_n$  and  $h_{n+1}$

can be shown to be

$$\begin{aligned} \frac{\partial |\varepsilon|}{\partial h_{n+1}} &= \frac{2|DD3| \left[ \frac{h_{n+1}(h_{n+1} + h_n)}{(h_{n+1}^2 + (h_{n+1} + h_n)(2h_{n+1} + h_n))} \right]}{(2h_{n+1} + h_n)^2} \\ \frac{\partial |\varepsilon|}{\partial h_n} &= \frac{|DD3| h_{n+1}^2 (h_{n+1} + h_n)(3h_{n+1} + h_n)}{(2h_{n+1} + h_n)^2} \end{aligned} \quad (10)$$

Both derivatives are positive, which implies that the LTE increases with both  $h_n$  and  $h_{n+1}$ , as they are expected.

In Fig. 2(b), it is assumed that the transient responses at  $t_1$  and  $t_2$  are already computed. As in a standard two-step integration method with LTE based time step control (e.g. (3, 9)), the first thread is launched to compute the circuit solution at  $t_3$  using the latest initial conditions at  $t_1$  and  $t_2$ . In parallel, a second thread is started to solve the solution at  $t_3'$ , which is in between  $t_2$  and  $t_3$ , using the same initial conditions. We refer to the computation of the  $t_3'$  circuit response as a backward step. Importantly, it shall be noted that since the LTE tolerance is satisfied at  $t_3$ , so is it at any placement of  $t_3'$  that is between  $t_2$  and  $t_3$ . Because of this, no accuracy issue is incurred. The work done at  $t_3'$  can be used in a meaningful way as follows. Upon the completion of the both threads, the solutions at  $t_3$  and  $t_3'$  will be used as the initial conditions for future time points. Compared with the serial transient simulation where the solutions at  $t_2$  and  $t_3$  are used as the initial conditions for the next time point, the availability of the  $t_3'$  solution in parallel backward pipelining reduces the value of  $h_n$  in the LTE based step control as in (9). Hence, backward pipelining leads to a larger next time step and advances the transient analysis faster along the time axis. The same two-thread pattern is repeated for solving the solutions at future time points such as  $t_4$  ( $t_4'$ ).

In practice, the location of  $t_3'$  (or  $t_4'$ ) shall be chosen to balance between efficiency and numerical stability. From an LTE point of view, placing  $t_3'$  closer to  $t_3$  allows for a larger next time step while making them too close to each other may introduce numerical problems. In our implementation, a damping factor  $\gamma$  is used to determine the location of  $t_3'$  such that  $t_3' - t_2 = \gamma \cdot (t_3 - t_2)$ ,  $0 < \gamma < 1$ .  $\gamma$  can be tuned experimentally to optimize the runtime while maintaining good numerical robustness. The double-threaded backward pipelining can be generalized for multi-step methods. We have developed a three-thread parallel scheme where two threads are launched to simultaneously compute the circuit solutions at two backward steps, which contribute to an improved future time step size in the three-step Gear's method.

## 3. FORWARD PIPELINING

The proposed parallel backward pipelining helps advance the transient analysis by providing better initial conditions in conjunction with multi-step integration methods. We further propose a forward scheme, which directly computes one or multiple future time points in parallel. Consider the situation shown in Fig. 3. It is assumed that the transient solutions at time points  $t_1$  and  $t_2$  are already computed. Using an LTE-based variable step-size two-step integration method, one thread may be used to compute the circuit solution at  $t_3$  using the solutions at  $t_1$  and  $t_2$  as the initial conditions. Again, as an naïve approach, one may attempt to use a second thread to compute the response at a time point  $t_4$  that is further down using the same initial conditions. While this seems to allow for two independent com-

putation tasks running in parallel, the solution obtained at  $t_4$  may not be trusted if the maximum permissible time step is already employed at  $t_3$ . In other words, the LTE tolerance may not be satisfied at  $t_4$ .

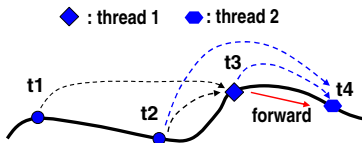


Figure 3: Double-threaded forward pipelining.

This problem is remedied in the proposed forward scheme, which is shown in Fig. 3. While the first thread is working at  $t_3$  using the  $t_1$  and  $t_2$  solutions as the initial conditions, a second thread is started to compute the solution at  $t_4$ . Here, the key difference is that the  $t_4$  solution is based on using the solutions at  $t_2$  and  $t_3$  as the initial conditions. Obviously, this creates data dependency between the two threads since the solution at  $t_3$  that is being computed by the first thread is not available yet. The data dependency and resulting issues are resolved as follows.

### 3.1 Prediction of Time Step Size

To start the work at  $t_4$  in parallel, the time step  $h_f = t_4 - t_3$  must be decided first.  $h_f$  also depends on the unknown circuit solution at  $t_3$ , say  $x(t_3)$ . To address this difficulty, an estimate of  $x(t_3)$  is quickly computed and used to launch the second thread. In particular, we employ Forward Euler (FE) rule to get the estimate, say  $\tilde{x}(t_3)$ . Since FE is explicit, such estimation can be done very efficiently. However, several complications arise and must be addressed. Using  $\tilde{x}(t_3)$  may lead to an overly optimistic time step size  $h_f$ . If this happens, the resulting solution at  $t_4$  does not satisfy the LTE tolerance and must be discarded. To reduce the chance of time step overestimation, in our approach a damping factor  $\beta$  ( $\beta < 1.0$ ) is introduced to scale down the estimated time step such that a more conservative time step is used:  $h_{f,damped} = \beta h_{f,FE}$ . If the LTE is still violated even with the use of damping upon the availability of the exact  $t_3$  solution, the predictive work done at  $t_4$  will have to be revoked, as detailed later in the paper.

### 3.2 Accuracy and Stability

Apart from the possibility of step size overestimation, the circuit solution computed in parallel at  $t_4$  using  $\tilde{x}(t_3)$  may not be accurate even if the time step is estimated conservatively. In this regard, the accuracy is guaranteed by one of the two inter-thread communication approaches in Fig. 4. As shown in Fig. 4 (a), in the coarse-grained approach at least one nonlinear Newton iteration is performed at  $t_4$  after the solution at  $t_3$  has fully converged. This guarantees that the  $t_4$  solution computed by thread 2 will converge to the exact value based upon the converged solution at  $t_3$ . It also implies one inter-thread communication in which thread 2 loads the converged  $x(t_3)$  computed by thread 1.

In the fine inter-thread communication, thread 2 more frequently updates  $\tilde{x}(t_3)$  as the convergence progress is being made by thread 1, as shown in Fig. 4 (b). The updated  $\tilde{x}(t_3)$  that is available at the end of each Newton iteration in thread 1 may be subsequently accessed to start the following Newton iteration in thread 2. Like before, upon the

completion of thread 1, the fully converged  $x(t_3)$  is loaded by thread 2 as the part of exact initial conditions to perform one or more Newton iterations to guarantee the accuracy of the  $t_4$  solution. Here, since the initial conditions are more frequently updated, thread 2 is made to converge faster, however, at the cost of somewhat higher inter-thread communication overhead. In our experiments, it is observed that the use of the finer grained communication scheme indeed further speeds up the transient simulation of large circuits, where the cost of Newton iterations is more dominant. Furthermore, it shall be noted that Forward Euler is only employed to estimate initial conditions and the simulation accuracy is strictly guaranteed at any time point using a stable integration method. Hence, the use of the FE based estimation does not incur any stability concern.

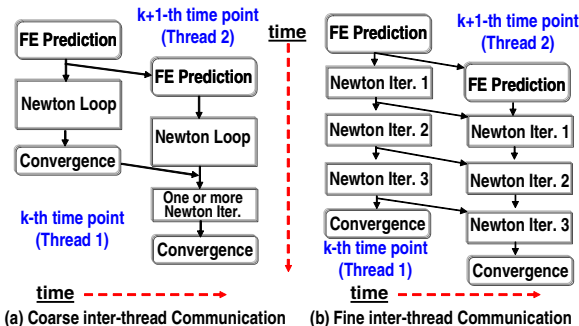


Figure 4: Fine and coarse grained inter-thread communications to guarantee the accuracy of the parallel forward scheme.

## 4. MULTI-THREADED WAVEPIPE AND THREAD SCHEDULING

The backward and forward schemes can be combined to create a variety of multi-threaded WavePipe implementations to utilize a larger number of processor cores, particularly when further combined with low-level parallel schemes such as parallel device model evaluations and matrix solvers. For the purpose of discussion, low-level parallelization is not considered. For a fixed number of threads/cores, multiple parallelizing schemes exist. For instance, a three-thread WavePipe may contain a thread that is allocated for standard variable-step size and multi-step integration (referred to as the base thread), one forward thread and one backward thread. Alternatively, the last two threads may be replaced by two forward threads. In the following, the threading scheduling issues in multi-threaded WavePipe are discussed using a four-thread (4T) implementation as an example, which contains one base thread, one backward thread, and two forward threads.

### 4.1 Thread Scheduling

As shown in Fig. 5, in this 4T implementation, starting from the two known circuit solutions as the initial conditions, the base thread T1 computes the circuit solution at the next time point according to a standard numerical integration method, in this case, Gear2. T1 first determines the time step and then computes an FE based estimation of the new solution. The FE estimation is used as an initial guess for the circuit response at the new time point. It is also

employed in forward pipelining to enable parallel predictive computing along the time axis, as described in Section 3.1. The second thread T2 is simultaneously launched to compute the circuit response according to backward pipelining. Upon the availability of the FE estimation computed by T1, the third thread T3 is started to facilitate the forward scheme. T3 also performs an FE based estimation for the circuit solution it will compute, which provides a basis to launch the second forward scheme thread, T4. The launching and completion of such four threads as a whole is referred to as a *thread scheduling cycle*. It shall be noted that within one scheduling cycle, T2 may complete slightly before T1. This is because that T2 works on a new time point that has a smaller time step than that of T1. The dependency between T1, T3 and T4 makes these three threads finish one after another.

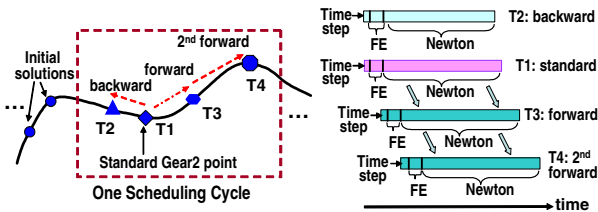


Figure 5: Four-thread waveform pipelining.

Once a scheduling cycle is completed, another cycle starts in the same fashion. As shown in Fig. 6 (a), if all the threads successfully complete in a scheduling cycle, the next cycle will start using the solutions computed by T3 and T4 as the initial solutions. However, as described in Section 3.1, it is possible to overestimate the time step in forward pipelining. If this happens, the work done by the corresponding thread is discarded. In Fig. 6 (b), it is assumed that T3 overestimates its time step. Hence, the solution computed by T3 does not satisfy the LTE tolerance and shall be discarded. Due to the data dependency between T3 and T4, the work done by T4 is discarded as well. In this case, the next scheduling cycle will use the solutions computed by T1 and T2 as the initial conditions.

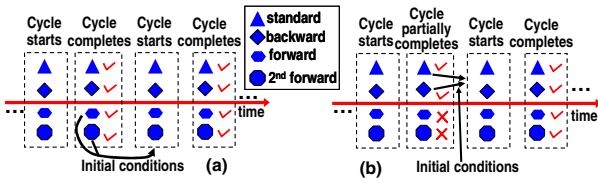


Figure 6: 4T waveform pipelining: a) without revoking of forward pipelining and b) with revoking of forward pipelining.

## 4.2 Scheduling Policies

The discussion above outlines a basic thread scheduling policy. In practice, multiple alternatives exist, which provide a basis for performance tuning of the parallel WavePipe implementation. For example, thread scheduling can be done with a finer granularity than a thread scheduling cycle. Note that within a cycle the four threads may complete at different times. By algorithm construction, forward pipelining

threads complete subsequently after the base thread. It is possible to launch new work immediately using any available thread without waiting the entire scheduling cycle to finish. However, this may or may not be beneficial depending on when the remaining running threads in the cycle can finish. The damping factor  $\beta$  introduced in Section 3.1 can be varied to modify the amount of conservativeness in the time step estimation in forward pipelining. A larger  $\beta$  value may help advance the transient analysis more rapidly, however, with a higher risk of revoking the work of forward threads.

In our implementation, multiple thread scheduling policies are implemented. The optimal policy together with the optimal values of various control parameters are experimentally selected to optimize the overall parallel simulation efficiency.

## 5. RESULTS

WavePipe is implemented in C/C++ using Pthreads library on a high-end shared-memory Linux server with four dual-core processors. To verify the performance of WavePipe on a variety of circuits, eight test circuits with distinguishing characteristics as shown in Table 1, are used in our experiments. Since the serial SPICE-like Backward Euler’s (BE) method and the serial two-step Gear’s method have comparable performances, only the results of the former are shown in Table 1, which are used as a reference to evaluate various parallel schemes. In Table 1, columns labeled as *Size*, *Points* and *Runtime* are the number of circuit unknowns, number of time points simulated and total transient simulation runtime, respectively. Note that there are nonlinear drivers in the two RLC mesh circuits.

Table 1: Statistics of test circuits and serial BE.

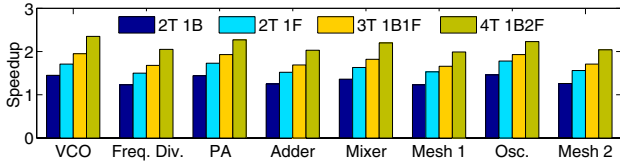
IDX	Circuit	Size	Points	Runtime(s)
1	VCO	20	90,545	39.54
2	Power Amplifier	8	118,426	31.27
3	DB Mixer	27	140,273	50.13
4	Ring Oscillator	61	115,973	217.47
5	Frequency Divider	17	45,693	18.85
6	Digital Adder	112	2,619	9.14
7	RLC Mesh 1	13,097	680	2,766.53
8	RLC Mesh 2	27,670	146	2,715.65

As described in Section 2.2, the three-thread parallel backward pipelining is implemented using the three-step Gear’s method. Compared with the two-thread two-step Gear based backward pipelining, up to 45% runtime speedup can be achieved. However, in practice, the stability issue has to be more carefully considered for higher order Gear’s methods. The results presented in the following are based upon the two-step Gear’s method.

In Table 2, we list the runtimes and speedups (w.r.t serial BE) of six coarse-grained WavePipe schemes, which are 2-thread backward pipelining, 2-thread forward pipelining, 3-thread one-backward-one-forward pipelining, 3-thread two-forward pipelining, 4-thread two-forward-one-backward pipelining, and 4-thread three-forward pipelining, respectively. Note that in all these schemes, there exists a base thread that implements the standard numerical integration. The average runtime speedups of the six schemes are 1.33x, 1.62x, 1.80x, 1.89x, 2.14x and 2.26x, respectively. In Fig.7, the runtime speedups of four of these six WavePipe schemes are visually presented. It is observed that the runtime scales almost linearly with the number of threads. In Fig. 8, we show a real-time profiling of the 3-thread one-forward-one-backward scheme running on an RLC mesh circuit. The runtimes of the three threads are break down to the follow-

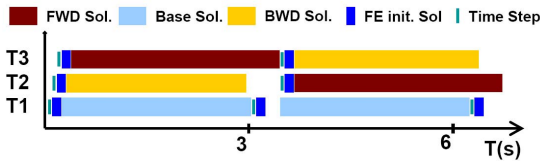
**Table 2: Runtime speedups of six coarse-grained WavePipe schemes: number of threads 2-4.**

IDX	2T 1-backward		2T 1-forward		3T 1-backward -1-forward		3T 2-forward		4T 1-backward -2-forward		4T 3-forward	
	T(s)	Speedup	T(s)	Speedup	T(s)	Speedup	T(s)	Speedup	T(s)	Speedup	T(s)	Speedup
1	27.3	1.45	23.1	1.71	20.3	1.95	19.6	2.02	16.8	2.35	16.1	2.45
2	21.7	1.44	18.1	1.73	16.2	1.93	15.4	2.03	13.8	2.27	13.2	2.36
3	36.9	1.36	30.8	1.63	27.6	1.82	26.3	1.90	22.7	2.20	21.6	2.32
4	149.3	1.46	121.9	1.78	112.4	1.93	107.2	2.03	94.7	2.23	91.0	2.39
5	15.3	1.23	12.6	1.50	11.2	1.68	10.7	1.77	9.2	2.05	8.7	2.16
6	7.3	1.25	6.0	1.52	5.4	1.69	5.1	1.79	4.5	2.03	4.2	2.18
7	2245.1	1.23	1814.6	1.53	1679.6	1.66	1559.0	1.78	1390.2	1.99	1324.6	2.09
8	2159.3	1.26	1742.2	1.56	1589.3	1.71	1487.4	1.82	1330.8	2.04	1265.4	2.15



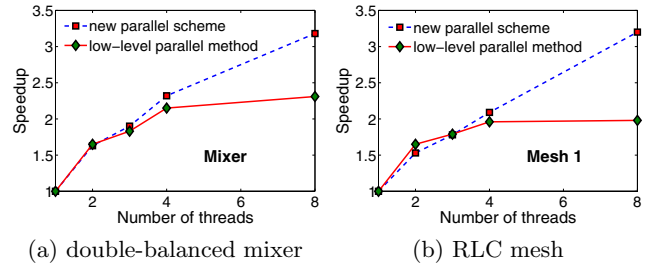
**Figure 7: Speedups of various WavePipe schemes.**

ing categories: time step computation, Forward Euler initial solution estimation, and remaining computation of one time step circuit response. The latter is further distinguished according to the mode of operation: base, forward (FWD) and and backward pipelining (BWD).



**Figure 8: Realtime thread profiling of the 3T one-forward-one-backward waveform pipelining.**

Next, we compare the proposed coarse-grained parallel WavePipe with the low-level scheme that bases upon parallel transistor device model evaluation and matrix solving. The public domain parallel matrix solver SuperLU [12] is employed. The comparison is made using a double-balanced mixer and an RLC mesh circuit in Fig. 9. When the number of threads is in between 2 and 4, our new schemes are completely based upon the proposed WavePipe. The 8-thread new scheme demonstrates the possibility of combining Wavepipe, in this case, the three-forward scheme, with parallel device model evaluation and matrix solver. In particular, within each mode of operation (one base mode, three forward modes), two threads are utilized to facilitate device model evaluation and matrix solving. When the number of threads varies from 2 to 4, WavePipe is comparable to the low-level parallel scheme. However, the runtime scaling of the low-level scheme already starts to saturate beyond four threads. The eight-thread parallel model evaluation/matrix solving scheme does not further improve the runtime. This clearly underscores the need to develop new application-level coarse-grained parallelizing avenues, which is the focus of this work, especially on massively parallel platforms. In contrast, our combined 8-thread scheme brings favorable speedups.



**Figure 9: Comparison between WavePipe and low-level parallel model evaluation/matrix solving.**

## 6. CONCLUSIONS

We propose a coarse-grained *Waveform Pipelining* approach to parallel transient circuit simulation. The backward and forward pipelining schemes allow us to exploit parallel computing along the time axis, hence offering new avenues to utilize application-level parallelism. Our experiments have demonstrated good efficiency factor of the proposed approach and its promising potential on parallel computing platforms with large numbers of processing cores.

## 7. REFERENCES

- [1] J. Friedrich et al. Design of the Power6<sup>TM</sup> microprocessor. In *ISSCC*, pages 96–97, February 2007.
- [2] U. Gajanan et al. An 8-core 64-thread 64b power-efficient SPARC SoC. In *ISSCC*, pages 108–109, February 2007.
- [3] S. Vangal et al. An 80-tile 1.28 TFLOPS network-on-chip in 65nm cmos. In *ISSCC*, pages 98–99, February 2007.
- [4] L. W. Nagel. *SPICE2: A Computer Program to Simulate Semiconductor Circuits*. ERL-M520, UC, Berkeley, USA, 1975.
- [5] E. Lelarasmee, A. E. Ruehli, and A. L. Sangiovanni-Vincentelli. The waveform relaxation method for time-domain analysis of large scale integrated circuits. *TCAD*, 1(3):131–145, July 1982.
- [6] J. K. White and A. Sangiovanni-Vincentelli. *Relaxation techniques for the simulation of VLSI circuits*. Kluwer Academic Publishers, Boston Norwell, MA, USA, 1987.
- [7] M. W. Reichelt, A. Lumsdaine, and J. K. White. Accelerated waveform methods for parallel transient simulation of semiconductor devices. In *ICCAD*, pages 270–274, Nov. 1993.
- [8] U. Wever and Q. Zheng. Parallel transient analysis for circuit simulation. In *HICSS*, pages 442–447. IEEE, January 1996.
- [9] L. T. Pillage, R. A. Rohrer, and C. Visweswariah. *Electronic Circuit and System Simulation Methods*. McGraw-Hill, 1995.
- [10] V. Litovski and M. Zwolinski. *VLSI Circuit Simulation and Optimization*. Chapman & Hall, New York, NY, USA, 1997.
- [11] H. Shichman. Integration system of a nonlinear network analysis program. *IEEE Trans. on Circuit Theory*, CT-17(3):378–386, August 1970.
- [12] J. Demmel, J. Gilbert, and X. Li. An asynchronous parallel supernodal algorithm for sparse gaussian elimination. *SIAM J. Matrix Analysis and Applications*, 20(4):915–952, 1999.