Parallel and Distributed VHDL Simulation*

Dragos Lungeanu
Deptartment of Computer Science
University of Iowa

C.J. Richard Shi
Department of Electrical Engineering
University of Washington

Abstract

This paper presents a methodology for parallel and distributed simulation of VHDL using the PDES (parallel discrete-event simulation) paradigm. To achieve better features and performance, some PDES protocols assume that simultaneous events may be processed in arbitrary order. We describe a solution of how to apply these algorithms to have a correct simulation of the distributed VHDL cycle, including the delta cycle. The solution is based on tie-breaking the simultaneous events using Lamport's logical clocks to causally order them according to the VHDL simulation cycle, and defining the VHDL virtual time as a pair of simulation physical time and cycle/phase logical time. The paper also shows how to use this method with a PDES protocol that relaxes the simulation of simultaneous events to arbitrary order, allowing the LPs to self-adapt to optimistic or conservative mode, without the lookahead requirement. The lookahead is application-dependent and for some systems may be zero or unknown. The parallel simulation of VHDL designs ranging from 5531 to 14704 LPs using these methods obtained a promising, almost linear speedup.

1. Introduction

Modern digital system design relies heavily on simulation to ensure design correctness and to maximize system performance. Simulation of very large scale integrated (VLSI) digital systems containing hundreds of millions of logic gates is time consuming, and has become a bottleneck in the design process [1].

In this paper, we present a new approach and its implementation to distributed and parallel simulation of digital VLSI systems described in VHDL [6]. Our approach is based on the parallel discrete-event simulation (PDES) paradigm. The foundation of our work is a general and novel lookahead-free self-adaptive optimistic and conservative synchronization protocol that allows maximal utilization of the inherent concurrency in digital systems [11]. In this paper, we describe how this protocol can be applied to digital VHDL simulation. More specifically, we present how VHDL signals and processes can be mapped to a PDES model as a distributed VHDL kernel. We propose to order simultaneous events based on Lamport's logical clocks [9], and define VHDL virtual time as a pair of simulation physical time and cycle/phase logical time. This leads to a distributed VHDL simulation cycle, which can handle VHDL delta cycles. A parallel VHDL compiler and simulator have been constructed and achieved almost linear speedup.

Previous research on parallel VHDL simulation was reported by [10, 13] based on optimized conservative synchronization. However, the method in [13] still relies on lookahead to avoid deadlock, and [10] assumes known static delays on each signal, and cannot handle delta cycles.

2. Parallel Discrete Event Simulation

We provide a brief overview of parallel discrete-event simulation (PDES). Excellent surveys can be found in [3, 4, 5]. The physical system under simulation is partitioned into entities that communicate via message passing. Each entity is modeled by a *logical process* (LP). The model of distributed simulation is a graph of logical processes exchanging *timestamp*ed *events* ($event@time$) over *channels*. Each LP has a *state* and a *simulate()* function. A *simulation step* of an LP calls the simulate() function with the next input event and current state, modifies the state and sends output events. The distributed simulation is correct if each LP processes its input events in chronological order of their timestamps (*local causality constraint* or *lcc*). The two major ways to ensure the lcc are: optimistic and conservative.

In the *conservative* methods [2], an LP **blocks** until it has a *safe* event to process. An event is safe if the LP will never receive another event with a lower timestamp. Blocking may cause *deadlock*, avoided or detected and recovered by global synchronization.

The *optimistic* methods [7] assume that all the events are safe. If there is a later event with a lower timestamp (*straggler*), it **rollbacks** by *time warping*. During the rollback the LP restores the state previous to the straggler and sends *negative events* to cancel all of the events sent during the wrong

*This work was sponsored by DARPA under grant No. F33615-96-1-5601

simulation. These negative events may cause rollback at their destinations. Rollback involves states and events saving and restoring, which may cause memory overflow. *Fossil collection* is the process in which unneeded memory cells are freed. Again, global synchronization is used to establish if a memory cell is old enough to be reused.

Global synchronization may be performed using *null message* or *global virtual time (gvt)* protocols. gvt is the smallest timestamp of an unprocessed event in the entire system. It is monotonically increasing over the simulation. A null message is an empty event with a timestamp. It is sent by an LP on the output channels to inform those LPs about the smallest timestamp of a future real event. This promise involves the knowledge of *lookahead*, a kind of delay from inputs to outputs. Lookahead is application-dependent and is sometimes zero, too expensive or impossible to compute. Without it, the null messages may not avoid deadlock.

2.1. Simultaneous Events in PDES

In a PDES model, events with the same timestamp can be generated. This may be a benefit if they are destined to different LPs. The order in which simultaneous events are processed by one LP may be irrelevant for some applications or may change dramatically the semantics for others. There are two main tie-breaking mechanisms for the simultaneous events used in PDES protocols: *arbitrary* in which an LP may process them in any order, and *user-consistent* in which the algorithm collects them all and passes the set to be ordered by the application. The second mechanism is more general, but requires overhead, which in many cases is unnecessary since some simultaneous events are still independent and may be processed in any order.

For optimistic protocols the overhead is extra rollback for events with equal timestamp. Rollback is already a big overhead needed to be reduced [12]. The conservative algorithms will block more until they are sure they will receive only events with strictly greater timestamps. This may lead to unavoidable deadlock for zero-delay cycles. For mixed protocols that combine optimistic and conservative methods, the user-consistent ordering cannot be guaranteed without a known lookahead [8].

Relaxing the assumption about simultaneous events to the arbitrary mechanism, we designed a mixed protocol that allows the LPs to dynamically self-adapt to optimistic or conservative behavior to find the best configuration. The protocol not only eliminates the user-consistent model overhead, but it is also lookahead-free. If the lookahead is available, it may be used to improve performance. When a conservative LP may receive events from an optimistic LP, it must be able to handle them without rollback. The protocol is described in detail in [11].

3. Distributed VHDL Kernel

After elaboration, the VHDL hierarchy is flattened into a graph of processes interconnected by signals [6]. This section describes how to map a post-elaboration VHDL model into a PDES model.

3.1. VHDL Signal

VHDL signals are not just simple channels. Complex semantics are associated with them. Signals may have multiple sources and multiple destinations. For each source there is a driver that holds a waveform on which transactions may be scheduled. In case of multiple drivers, a resolution function is defined to resolve the effective value of the signal.

In a distributed system the communication is based on message passing and there is no shared memory to store the signals. Since many VHDL processes may use the value of the same signal, we face a distributed memory problem.

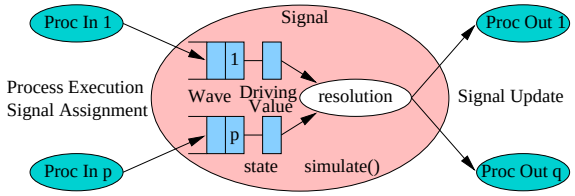

Figure 1. VHDL Signal LP.

Our solution is to map each signal as a PDES LP (Fig. 1). The signal LP will hold in the state one driver for each source and will broadcast the new effective value to each VHDL process that uses it. This value is computed by applying the resolution function on the driving values of each input driver. The $simulate()$ function also takes care of updating the waveforms and of scheduling transactions for future driving values.

3.2. VHDL Process

The basic computational block of a VHDL design is the process statement which models the behavior in terms of sequential statements. VHDL processes map naturally into PDES LPs (Fig. 2).

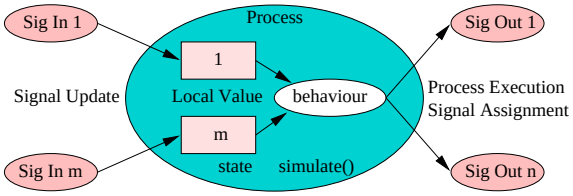

Figure 2. VHDL Process LP.

The state contains the process variables and local copies of the effective values for each input signal used in sensitivity lists or in any expression.

The $simulate()$ function is called each time there is an event for the LP. The event may be external from an input signal that changed its effective value, or may be internal from the LP itself to trigger the execution of the sequential statement part of the VHDL process.

An external event will update the local copy of the input signal and will schedule an internal event if the process is sensitive to the signal or if the wait condition (if any) becomes true. In this case, any pending timeout event will be canceled.

An internal event will cause the process execution until the next wait statement. If there is a timeout clause, another internal event is scheduled. During the execution, the process may send events to output signals as a result of signal assignment statements.

3.3. Distributed VHDL Simulation Cycle

In a PDES model each LP executes in an asynchronous, independent manner with different local simulation times. Various protocols are used to synchronize them and to ensure a correct simulation. The synchronization is based on using timestamps for the events, and on each LP to process its events in chronological order of the timestamps (lcc). To achieve better features and performance, some PDES algorithms assume that simultaneous events may be simulated in arbitrary order. This assumption may modify the semantics of the VHDL simulation, leading to incorrect results in cases of delta cycles, timeouts, signals with multiple simultaneous transactions or processes with multiple simultaneous input signals updates.

We propose a tie-breaking scheme for the problematic simultaneous events to conform to the VHDL simulation cycle. The solution is based on appending an extra field to the VHDL simulation time, which specifies the VHDL cycle/phase number. So the virtual time is a pair of physical simulation time and logical cycle/phase number:

$$vt = (pt, lt)$$

with the order relation $vt_1 < vt_2$ true if and only if:

$vt_1.pt < vt_2.pt$ or ($vt_1.pt = vt_2.pt$ and $vt_1.lt < vt_2.lt$)

This solution is similar to Lamport's logical clocks [9] in the sense that it defines an ordering on the events based on their causality. Using this notion of simulation virtual time, the distributed simulation cycle has the following phases (Fig. 3):

Signal: Signal Assign: $vt = (pt = now, lt = 3k)$ Due to signal assignments in the **Process: Process Execution** phase, the process LP may send events to output signals. The signal will update the corresponding waveform conform to the delay mechanism, will add new transactions,

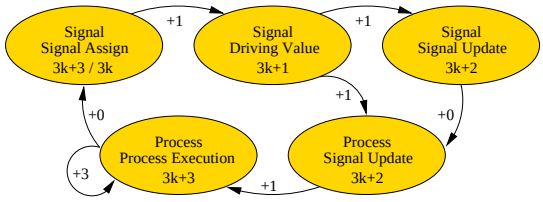

Figure 3. Distributed VHDL Cycle.

and for each transaction will schedule an internal event for the next **Signal: Driving Value** phase with timestamp $ts = (now, lt + 1)$ if $after = 0$, or $ts = (now + after, 1)$ if $after > 0$.

Signal: Driving Value: $vt = (pt = now, lt = 3k + 1)$ The next transaction in any input waveform is performed. It will update the corresponding driving value and schedule an internal event for the next **Signal: Signal Update** phase with timestamp $ts = (now, lt + 1)$ if the signal is resolved. The reason is that another driver may have pending a transaction at the same time, so the resolution function must be applied after all these simultaneous transactions. If the signal has only one source, then the driving value becomes the effective value and if changed, it is broadcasted to all fan-out processes by sending events with timestamp $ts = (now, lt + 1)$.

Signal: Signal Update: $vt = (pt = now, lt = 3k + 2)$ If the signal is resolved, the resolution function is applied for all driving values to compute the effective value, which is sent to all output processes through events with timestamp $ts = (now, lt)$.

Process: Signal Update: $vt = (pt = now, lt = 3k + 2)$ The VHDL process LP receives new effective values for the input signals and updates the corresponding local values. If the process is sensitive to the signal, or if the wait condition becomes true, then an internal event is scheduled for the next **Process: Process Execution** phase with timestamp $ts = (now, lt+1)$, and the pending (if any) internal timeout event is canceled. The reason for incrementing the logical time for the next phase is to be sure that all simultaneous signal updates take place before process execution. However, the order of simultaneous signal updates is irrelevant.

Process: Process Execution: $vt = (pt = now, lt = 3k + 3)$ The VHDL process loop is resumed due to either signal change or timeout event by calling a *run()* virtual method. The process may send events to output signals as a result of signal assignments and will suspend on the next wait statement. If a timeout is specified in the wait call, an internal event is scheduled for this phase in the next cycle with timestamp $ts = (now, lt + 3)$ if $timeout = 0$, or $ts = (now + timeout, 3)$ if $timeout > 0$.

3.4. Remarks

Mapping both VHDL processes and signals to PDES LPs will create a bi-partite graph. Partitioning this graph for balancing the load and minimizing the communication may take advantage of this particular topology, leading to a faster and better solution.

A possible alternative to single-source signals is to map them in the same LP as the driving VHDL process to reduce the overall number of LPs. However, this will complicate the VHDL process LP $simulate()$ function and will destroy the bi-partite property of the graph.

4. Implementation and Results

The lookahead-free, dynamic, mixed PDES protocol was implemented in C++, using MPI or TCP/IP sockets for communication. We have also developed a VHDL to C++ translator, and a VHDL kernel library to support signals, processes and VHDL statements not directly translatable to C++, like signal assignment and wait. Both VHDL process LP and VHDL signal LP classes are derived from the LP class, and implement the above distributed simulation cycle in the virtual *simulate()* method. For each VHDL process there is a C++ class derived from a VHDL process LP whose *run()* virtual function is given by the VHDL process sequential statement part and called in the **Process Execution** phase.

The circuits in Figs. 5, 7, and 9 were described in VHDL at the behavioral and gate level, and simulated on an SGI Challenge parallel machine with 16 processors, using 4 different configurations: all LPs optimistic, all conservative, registers conservative and combinational part optimistic, all dynamic. All simulations were verified to be correct. The sizes of the circuits and the speedups relative to the 1 processor execution (improved for sequential simulation) are illustrated in Figs. 6, 8, 10. Despite the naive partitioning used (equal number of LPs to each processor), which caused occasional dips in the curves, the speedups are linear and show a good potential for parallel VHDL simulation.

We observe that in general the optimistic configuration is very suitable for digital simulation. Unfortunately, it demands huge amounts of memory, proportional to the number of processors. Heavy-state processes cannot save their state, so they must run conservatively. The conservative configuration is better when there is a large number of simultaneous events (Fig. 6). Since we did not use the lookahead, the overhead of null messages was disabled.

The mixed configuration in which synchronous components are mapped as conservative and asynchronous ones as optimistic worked very well for most of the cases, better than all optimistic or all conservative configurations. We based our heuristic on the fact that the clock signal is very persistent and in most of the cases ready before other inputs are stable. On the other hand, asynchronous events follow a data-flow path, and are usually safe.

The most impressive results come from the dynamic synchronization, which is able to follow closely the best configuration (out of 4 tried) or find it automatically. For the gate level simulation of DCT processor, the speedup for the self-adapting dynamic configuration is twice the speedup of other configurations (Fig. 10).

To compare the arbitrary and the user-consistent models, we modified the protocol to support the second one for all LPs conservative or all LPs optimistic. We also had to turn on the lookahead support, since the user-consistent model for conservative configuration will block without it. The running times in seconds on 4 processors are shown in Fig. 4. The overhead of user model itself is small, but for light VHDL LPs, the main overhead is the need of lookahead (+la) and the null messages to propagate it.

Circuit	Conservative				Optimistic	
	arbitrary		user		arbitrary	user
	-la	+la	+la	-la	-la	-la
FSM	17.1	27.2	27.6	∞	20.8	22.0
IIR	22.6	43.5	46.4	∞	21.4	22.6
DCT	49.2	250	289	∞	38.4	41.4

Figure 4. Arbitrary vs. User-Consistent.

5. Conclusion

This paper presented a methodology for distributed VHDL simulation, with solutions for signals distributed memory problem and distributed VHDL cycle, by introducing a tie-breaking mechanism for problematic simultaneous events, based on extending the VHDL physical simulation time with the cycle/phase logical number to causally order them. This method avoids the overhead of user-consistent PDES protocols that rollback or block too much to order even independent simultaneous events. With this method we could use a PDES protocol that relaxes the assumption about simultaneous events to arbitrary order, achieving features that were otherwise impossible with the user-consistent model. The protocol is application-independent, working without knowing the lookahead, allowing the LPs to self-adapt to either optimistic or conservative mode for the best configuration. This combination works for any VHDL circuit, including delta cycles, making it very convenient and a strong candidate for automatic translation for parallel simulation of VHDL. The results of the simulation of large real VHDL circuits have shown promising speedups.

References

[1] R. D. Chamberlain. Parallel logic simulation of VLSI systems. *Proc. 32rd IEEE/ACM DAC*, p. 139, Jun.1995.

[2] K. M. Chandy and J. Misra. Asynchronous distributed simulation via a sequence of parallel computations. *Comm. of the ACM*, Vol. 24, No. 11, p. 198, Nov.1981.

[3] A. Ferscha. Parallel and distributed simulation of discrete event systems. *Parallel and Distributed Computing Handbook*, McGraw-Hill, 1995.

[4] R. M. Fujimoto. Parallel discrete event simulation, *Comm. of the ACM*, Vol. 33, No.10, p. 30, Oct.1990

[5] M. Bailey, J.V.Jr. Briner and R.D. Chamberlain. Parallel logic simulation of VLSI systems, *ACM Computing Surveys*, Vol. 26, No. 3, p. 255, Sep.1994

[6] IEEE. IEEE standard VHDL language reference manual, *IEEE Std. 1076-1993*, 1994

[7] D. A. Jefferson. Virtual time. *ACM TOPLAS*, Vol. 7, No. 3, p. 404, Jul.1985.

[8] V.Jha and R. Bagrodia. A unified framework for conservative and optimistic distributed simulation. *Proc. 8th. PADS*, 1994

[9] L. Lamport. Time, clocks, and the ordering of events in a distributed system, *Comm. of the ACM*, 21(7), p. 558, Jul.1978

[10] E. Naroska. Parallel VHDL simulation, *Proc. DATE*, p. 159, 1998

[11] D. Lungeanu and C.J. Richard Shi. Distributed simulation of VLSI circuits via lookahead-free self-adaptive optimistic and conservative synchronization, *Proc. ICCAD*, p. 500, 1999

[12] S. Schmerler et al. Advanced optimistic approaches in logic simulation, *Proc. DATE*, p. 362, 1998

[13] P.A. Walker and S. Ghosh. Asynchronous, distributed event driven simulation algorithm for execution of VHDL on parallel processors, *Proc. 32rd IEEE/ACM DAC*, p. 144, Jun.1995.

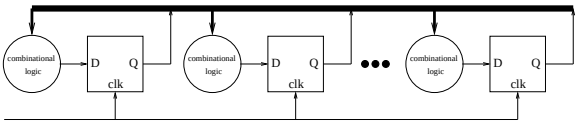

Figure 5. Finite State Machine (FSM).

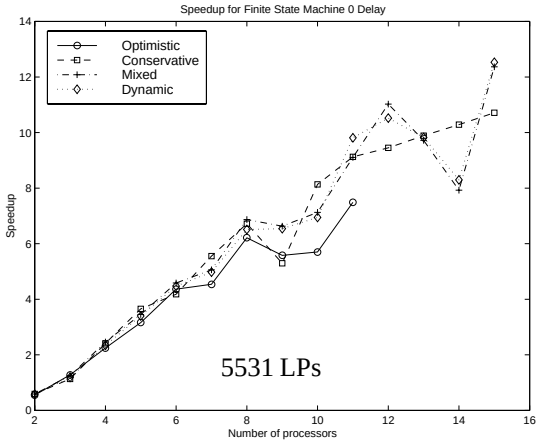

Figure 6. Speedup for FSM (0 Delay).

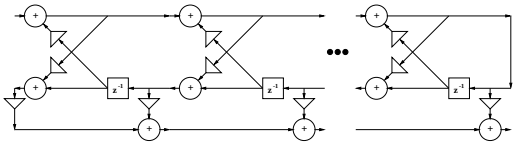

Figure 7. Gray-Markel Cascaded Lattice IIR Filter.

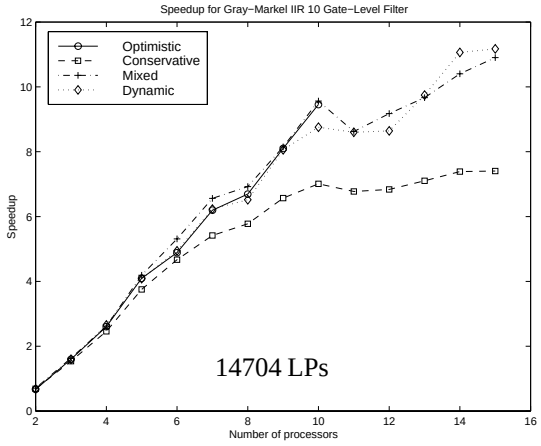

Figure 8. Speedup for IIR Filter (Gate).

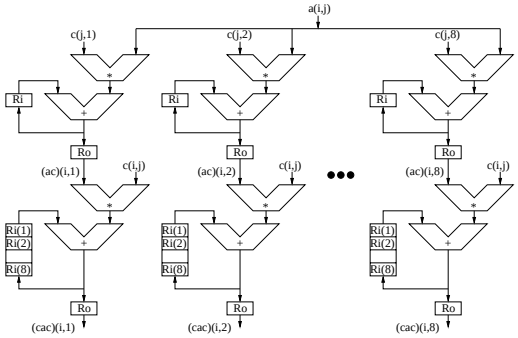

Figure 9. Discrete Cosine Transform Processor.

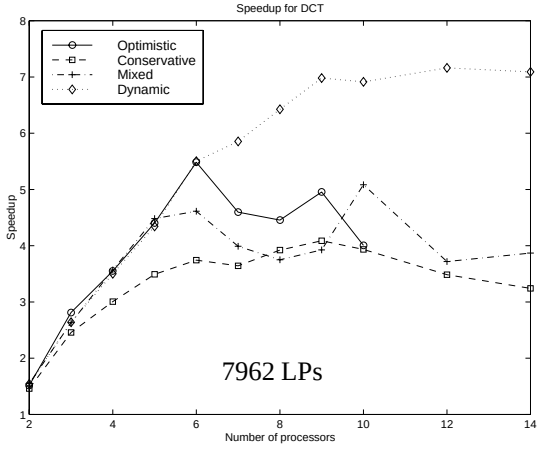

Figure 10. Speedup for DCT Processor (Gate).