Parallel VHDL Simulation

E. Naroska
Computer Engineering Institute
University of Dortmund
44221 Dortmund, Germany

Abstract

In this paper we evaluate parallel VHDL simulation based on conservative parallel discrete event simulation (conservative PDES) algorithms. We focus on a conservative simulation algorithm based on critical and external distances. This algorithm exploits the interconnection structure within the simulation model to increase parallelism. Further, a general method is introduced to automatically transform a VHDL model into a PDES model. Additionally, we suggest a method to further optimize parallel simulation performance. Finally, our first simulation results on a IBM parallel computer are presented. While these results are not sufficient for a general evaluation they show that a good speedup can be obtained.

1 Introduction

VHDL [3] is a formal programming language intended for use in all phases of digital electronic design. As the complexity of digital designs continuously increases, validation by simulation becomes the most time and resource consuming task. To deal with this problem parallel VHDL simulation on a parallel computer or a network of workstations has been suggested. This approach is based on the parallel discrete event simulation (PDES) paradigm and promises to reduce the simulation time by distributing the simulation workload on many nodes of a parallel computer.

Previous research in PDES has resulted in two main simulation approaches, the conservative and the optimistic one. In the conservative approach all events of a process P are executed in the order given by their time-stamps [5], i.e. the chronological order of events is preserved at each process. Conservative algorithms can be further separated into blocking and non-blocking ones. A blocking approach has been introduced for instance by Chandy and Misra [2]. Soule and Gupta [8] reported that these algorithms tend to spend a lot of computation time for deadlock detection, deadlock recovery and additional process computation when applied to simulation of digital circuits. Non-blocking algorithms prevent deadlocks but require often additional computation and communication overhead [8].

Alternatively, the optimistic approach allows the processing of an event at a process without guaranteeing that no other event with a smaller time-stamp may arrive at the same process [4]. If the chronological order is violated, the simulation must be rolled back in time to a safe state. Therefore, an optimistic simulation algorithm requires a strategy for backing up the simulation. This results in additional memory requirements which may be unacceptable for large VLSI simulation tasks.

For our VHDL simulation we choose a conservative non-blocking approach. Similar to [1, 7] our approach is based on distances between processes. However, in order to adapt to parallel computers with message passing we avoid all barrier synchronization. Instead, a special kind of state messages (null-messages) for asynchronous *node* synchronization similar to [7] are used.

2 PDES based on critical and external distances

A discrete-event simulation is represented by a weighted di-graph $G(P, W, l)$ with $W \subseteq P \times P$ and $l : W \rightarrow \mathbb{N}_0^+$. Intuitively, $p \in P$ and $w = (p_i, p_j) \in W$ denote a logical process and a link between two logical processes, respectively. During the simulation a message q, called an event, may be sent across a link (p_i, p_j). Each event q consists of a time stamp $t(q)$ and a value.

Upon receiving an event q on link (p_i, p_j) the destination process $p_{dest}(q) = p_j$ may change its internal state and may also send events on its output links. The virtual time $t(p_j)$ of p_j is set to $t(q)$. The time stamp $t(q')$ of any outgoing event q' on link (p_j, p_k) is at least $t(p_j) + l(p_j, p_k)$. Thus, the weight $l(w) \geq 0$ of a link $w = (p_i, p_j)$ denotes the lower bound of the event delay on this link.

Consequently, the earliest time, at which a process p_i may influence the state of a process p_j, is given by $t(p_i) + d(p_i, p_j)$ where $d(p_i, p_j)$ is the length of the shortest path in G from p_i to p_j. As the simulation is causal, $t(p_i)$ is not allowed to decrease. Hence, an appropriate scheduling method must guarantee that no event e_x may arrive at p_i with $t(e_x) \leq t(p_i)$.

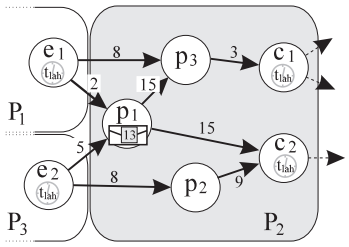
Prior to parallel simulation a part of the model (partition) is assigned to each simulation processor in a step called partitioning. We assume that each processor is assigned a single partition which consists of a large amount of processes. As we concentrate on the synchronization algorithms in this paper we further assume that an appropriate partitioning of the simulation models is given.

Lookahead is a very important part of PDES. It is an estimation on the future behavior of a process and is used to determine event dependencies during the simulation. A lookahead value $t_{lah}(p)$ of a process p states that any yet unprocessed event q will have an time stamp greater equal to $t_{lah}(p)$.

Asynchronous conservative simulation algorithms usually execute in two main steps. First, event dependencies caused by events located on different computation nodes (partitions) are determined in a step called event state analysis. This analysis is based on lookahead information provided by other partitions. As a result the state of an event becomes either ready or blocked. Usually, event state analysis is done by simply comparing the event time stamp with the minimal lookahead (channel) time of all incoming links of the partition. An event is blocked if the time stamp is greater or equal to that value. While this method produces only moderate computation overhead the interconnections of the processes *within* a partition have no effect on event state analysis. As a result, events may be blocked by links which in fact have no influence on these events. During the second step all ready events are executed in time stamp order.

Our scheduling algorithm is conservative and asynchronous as well. However, it uses so called critical and external distances which represent the communication structure of the processes located on a partition in order to determine the state of the events and the lookahead [6]. As a result, mutually independent events of a *partition* are recognized and executed if they cannot be affected by an event located on another partition i.e. if they are ready. Blocked events are not executed until they become ready. This requires recalculating the state of blocked events when new lookahead information arrives.

In the Fig. 1 three partitions P_1 to P_3 of a simulation model are shown. The logical processes are represented by circles. The processes labeled c_1, c_2, e_1 and e_2 are normal processes like p_1 to p_3 but have a special meaning to partition P_2. The (external) processes e_i form the external process set E_2 of P_2. They are not located on P_2 and have at least one outgoing link connected with a process of P_2. On the other hand the (critical) processes c_i form the critical process set C_2 of P_2. These processes belong to P_2 and are elements of at least one external process set of another partition.



critical distances

	p_1	p_2	p_3	c_1	c_2	e_1	e_2
c_1	18	-	3	0	-	11	23
c_2	15	9	-	-	0	17	17

external distances

	p_1	p_2	p_3	p_4	p_5
e_1	2	-	8	11	17
e_2	5	8	20	23	17

Figure 1: Event state analysis and lookahead calculation based on distances

Consider an event q with a time stamp $t(q)$ which resides at process $p_{dest}(q)$ located on a partition P_n. To determine the state of q, the influence of each external process of P_n on q is determined by evaluating the expression

$$t(q) < t_{lah}(p_e) + d(p_e, p_{dest}(q)) \; \forall \, p_e \in E_n. \qquad (1)$$

If at least one in-equation evaluates to false, the event is blocked, otherwise it is ready. Ready events may now be executed e.g. in time stamp order. This will preserve the dependencies between the local (blocked as well as ready) events.

Lookahead calculation is done in a similar way. To estimate the lookahead for a critical process p_c of a partition P_n the expression

$$t_{lah}(p_c) = \min\{\{t_{lah}(p_x) + d(p_x, p_c) \mid p_x \in E_n\}, \{t(q) + d(p_{dest}(q), p_c) \mid q \in Q_n\}\} \qquad (2)$$

is calculated, where Q_n denotes the local event set of partition P_n. The distances $d(p_i, c_i)$, $c_i \in C_n$ are called critical distances. The new lookahead for process p_c may now be sent to other partitions asynchronously. Because the link weights are static, the distances $d(p_i, p_j)$ are static as well and may be stored into a critical respectively external distance table as shown in Fig. 1.

3 Model transformation

Parallel VHDL simulation is based on the parallel discrete event simulation paradigm (PDES). Because there is a wide variety of simulation algorithms for common PDES problems, it is appropriate to transform the VHDL model into the common PDES model.

The first problem occurring during the transformation process is the handling of VHDL signals. Signals are used

in VHDL for inter process communication. Processes may read from and write on signals. Additionally, processes may be executed on a change of a signal value. In general, there are three different ways how to deal with signals. The first solution is to combine them with the VHDL processes, which are writing values to them, to form a hybrid node. Unfortunately, this may reduce lookahead if a process writes to several signals. For instance, if a process p_1 has write connections to two signals s_1 and s_2, these nodes are combined to one single hybrid node. As a result, transactions created for e.g. s_1 are located on that hybrid node. Therefore, the lookahead algorithm cannot determine that these transactions have no affect on s_2.

Alternatively, signals may be combined with the VHDL processes which are reading from them. However, if a signal is read by several processes then transactions for that signal must be copied and distributed to all corresponding hybrid nodes. Unfortunately, this may result in additional computation overhead, if the transaction of the signal does not produce any event i.e. does not change the value of the signal. In this case this transactions are first inserted and then all removed from the system, separately. Moreover, most transactions in VHDL do not produce any events.

On the other hand using distinct signal and process nodes avoids these problems but requires additional memory and computing overhead to handle the signal nodes. Fortunately, this overhead may be reduced due to the special properties of the signal nodes.

Every signal is represented by a driver and a reader node corresponding to the driver and reader part of a VHDL signal [3]. In addition each resolved signal is associated with a resolution node representing the resolution function. Port map connections using conversion functions are handled by distinct conversion nodes. The different nodes are connected by links modeling the information flow, which is necessary to obtain the reader values for the signals. To handle the different VHDL delay mechanisms, each link e is associated with a weight $l(e)$ consisting of three time values $(adist(l(e)), cdist(l(e)), wdist(l(e)))$, which are described separately:

- The activation weight $adist(l(e))$ represents the smallest time difference between an event on the source node and an event which may *activate* the destination node.
- The clearing weight $cdist(l(e))$ is connected with inertial delays. Executing an event q on a source node p_s may remove all events on the destination node p_d with a time stamp greater or equal than $t(q) + cdist(l(p_s, p_d))$.
- The write weight $wdist(l(e))$ represents the smallest time difference between an event of the source node and a resulting event on the destination node.

With this technique the PDES mechanisms can be used to provide the signal update mechanisms as defined in the VHDL standard [3].

Next, we address how to analyze the possible interactions between VHDL processes and signals. After simulation startup all links between processes and signals are determined. If a VHDL process node may read from a signal node or may be sensitive on that node, a link leading from the reader node of that signal to the process node is inserted into the model. The weight of this link is set to $(0, 0, 0)$ if the process may be sensitive on the signal or to $(0, 0, \infty)$, otherwise. If a process includes statements which write on a signal, then a link e leading from the process to the driver of the signal node is inserted. The weight of that link depends on the signal write statements included within the process code[1]:

- $cdist(e)$ is set to the minimal difference between the delay and the corresponding reject time of all signal assignment statements related to that signal. If this minimum is 0 then $cdist(e)$ is set to $1d$.
- $wdist(e)$ is set to the minimal value of the delay of all signal assignment statements related to the signal or $1d$ if this minimum is 0.
- $adist(e)$ is set to $wdist(e)$.

Fig. 2 shows an example VHDL model consisting of 2 processes and Fig. 3 the corresponding transformed model. Some optimizations may be obtained by combining nodes of the model. This reduces the amount of nodes and as a result the simulation overhead and memory consumption. However, a detailed description of the transformation and the optimization rules is beyond the scope of this paper.

4 Optimization for parallel VHDL simulation

Usually, digital circuits include a large number of flipflops which create new events on their outputs only if their clock input changes. In contrast, events on the data inputs cannot activate the flipflop. However, during the transformation of the VHDL model into a PDES graph, all communication channels must be introduced as edges into the PDES graph regardless whether they may activate a process or represent pure data dependencies only. To distinguish between data and activation dependencies, the activation delay $adist$ of all pure data dependency links is set to ∞ as shown in Section 3. Fig. 4 shows a digital circuit consisting of 4 flip-flops and 3 gates. The flip-flops have

[1] The corresponding weight value is set to $1d$ if the minimal differences cannot be determined after simulation startup.

```
-- a, b, and c are signals of type INTEGER
p1: PROCESS            -- definition of process p1
BEGIN
    a <= c + 1 AFTER 2 ns;
    b <= a + b AFTER 10 ns;
    IF a = b THEN
        WAIT ON a;          -- wait statement #1
    ELSE
        WAIT ON b;          -- wait statement #2
    END IF;
END PROCESS;

p2: PROCESS (b)        -- definition of process p2
BEGIN
    c <= TRANSPORT b - c AFTER 30 ns;
END PROCESS;
```

Figure 2: VHDL model with 2 processes

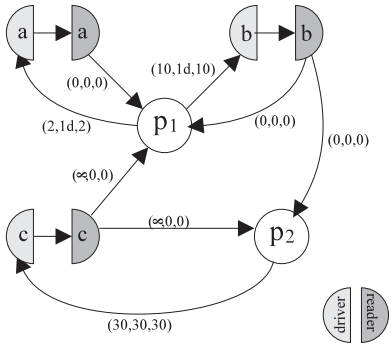


Figure 3: Transformed VHDL model (all link weights not shown are $(0, 0, 0)$)

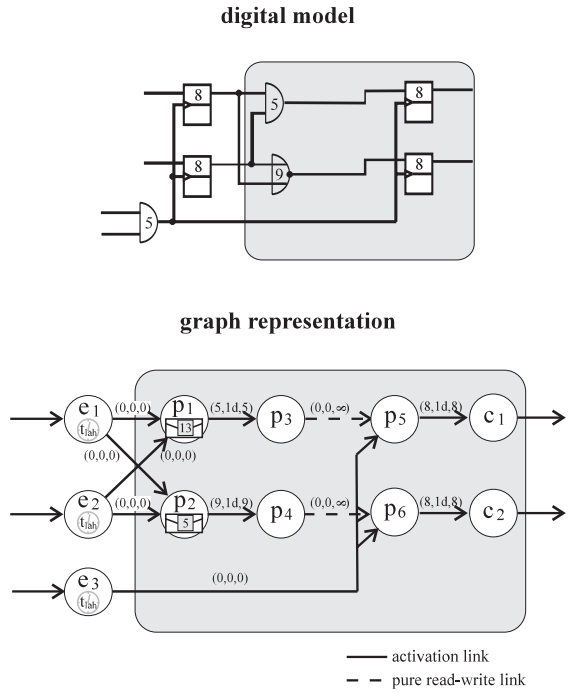


Figure 4: An example model and corresponding PDES graph

an inertial delay of 8 and the gates of 5, respectively 9 time units. Fig. 4 also displays the corresponding PDES model of the grey shaded partition. To calculate the external distances of the model we must use the $cdist$-values of the weights to take the delay model of VHDL in consideration. However, determining the critical distances may be based on the activation weights $adist$ only. As a consequence the lookahead of e.g. c_2 is only dependent on the events submitted by e_3 and p_6 ($d(e_3, c_2) = 8$, $d(p_6, c_2) = 8$). The other nodes (p_1 to p_5, e_1 and e_2) may also produce events for c_2 directly or indirectly, but none of them can activate c_2 i.e. the corresponding critical distances are equal to ∞.

Using the activation weights for lookahead calculation we obtain two advantages. First, the lookahead value usually can be increased compared to lookahead calculations based on the write values $wdist$ of the weights. Second, less operations are required to determine the lookahead, as less events and lookahead values can affect the critical processes. For example in Fig. 4 event located on processes e_1, e_2 and p_1 to p_4 cannot affect c_1 or c_2 and may be omitted during lookahead calculation.

5 Simulation results

We have implemented a parallel VHDL compiler/simulator prototype based on the above described concepts. The system is running on an IBM SP2

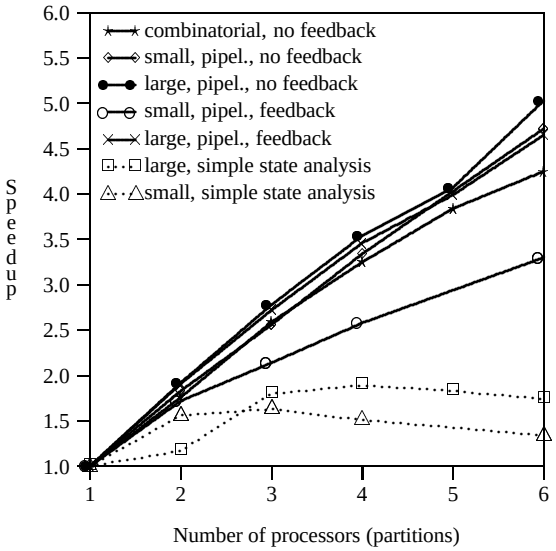


Figure 5: Speedup of the large and the small model

parallel computer and has been used to simulate various versions of a multiplier at gate level (latches and combinational logic) consisting of 10,771 (large model) and 2,699 (small model) VHDL processes. Additionally, we implemented a version of each model which included a feedback path from the outputs of the models back to their inputs. Finally, we simulated a non pipelined multiplier (combinational model) consisting of 6164 VHDL processes. All models were manually partitioned.

In Fig. 5 the speedup curves for the models on the parallel computer are shown. For almost all models we got good speedup curves. Only the small pipelined multiplier with feedback fails to achieve speedup above 3.5 due to the small amount of VHDL processes present in the model. Further, to compare state analysis based on external distances with simple approaches we included two additional speedup curves (“simple state analysis”, dotted curves) for a large and a small pipelined model with feedback using a simplified state analysis algorithm. Simple state analysis in our experiments was done by combining the external processes to form two *virtual* external processes. Event state calculation was then based on the external distances to this virtual external processes. This approach is similar to the algorithms which compare the event time stamp with the minimal incoming channel times directly. As shown in the Figure the simple approach fails to achieve a speedup value greater then 2.0 respectively 1.6.

6 Summary and conclusion

In this paper we introduced methods to transform VHDL models into PDES models. This methods allow the use of different PDES algorithms for parallel VHDL simulation. We used an algorithm based on critical and external distances to synchronize the partitions of the simulation computer. Further, a optimization technique has been discussed which helps to exploit the parallelism within a VHDL simulation model.

A prototype compiler/simulator has been used to obtain some experimental results for several VHDL models. While these results are not sufficient for a general evaluation they show that a good speedup can be obtained.

References

[1] R. Ayani and H. Rajaei, Parallel Simulation Based on Conservative Time Windows: a Performance Study, Concurrency: Practice and Experience, vol.6(2), pp.119-142, April 1994

[2] K.M. Chandy and J. Misra, Asynchronous Distributed Simulation via a Sequence of Parallel Computations, Comm. of the ACM, vol.24, no.11, pp.198-206, Apr 1981

[3] IEEE, IEEE Standard VHDL Language Reference Manual, IEEE Std 1076-1993, 1994

[4] D.R. Jefferson, Virtual Time, ACM Trans. on Prog. Lang. and Sys., vol.7, no.3, pp.404-425, Jul 1985

[5] J. Misra, Distributed Discrete-Event Simulation, Computing Surveys, vol.18, no.1, pp.39-65, Mar 1986

[6] E. Naroska and U. Schwiegelshohn, A new Scheduling Method for Parallel Discrete-Event Simulation, In Proc. of the 2nd Int. Euro-Par Conf., Lyon, vol II, pp.582-593, Aug 1996

[7] D. Nicol, Parallel Discrete-Event Simulation of FCFS Stochastic Queuing Networks, SIGPLAN Not., pp.124-137, Sep 1988

[8] L. Soulé and A. Gupta, An Evaluation of the Chandy-Misra-Bryant Algorithm for Digital Logic Simulation, ACM Trans. on Modeling and Comp. Sim., vol.1, no.4, pp.308-347, Oct 1991