

PARALLEL SIMULATION: SELF-EXTRACTION OF THE NATURAL PARALLELISM OF VHDL MODELS

Y.HERVE

ENSPS/LSIT, 7 rue de l'université
67000 STRASBOURG - FRANCE
Tel:(33) 88 35 80 84 - Fax:(33) 88 35 31 76
Email:yann@sobel.u-strasbg.fr

ABSTRACT

Performance limitations of distributed simulations are due to the target machine, the simulation algorithm and the natural parallelism of the model. In order to build a set of well known benchmarks, we propose here an instrumentation method of VHDL models. This method allows extraction of natural parallelism during a sequential simulation.

1. INTRODUCTION AND AIMS

The VHDL language is more and more used. Its multi-level capabilities allow description of very complex systems but the time of simulation become prohibitive [1]. One trend is to propose some VHDL parallel simulators on a work-station farm with a local area network or on a parallel MIMD machine [4][5][6].

Limitations of parallel simulations are due to three kinds of parameters:

- architecture of the machine (type and number of processors, communication network, shared memory,...)
- parallel simulation algorithm ([3] for a survey) (centralized event list, strong synchronization, conservative or optimistic approach (CHANDY-MISRA or TIME-WARPING),...)
- natural parallelism of the model.

In this paper, we will present a method to characterize limitations imposed by the model itself. The natural parallelism will take in account the experience (sequence of input vectors), relative complexity of processus in the model and will be independent from simulation machine or from specific algorithm (sequential or parallel) [7][8].

After a presentation of the general methodology, we will outline the algorithm and its application to VHDL models in multi-level approach. We will raise some problems and limitations in the instrumentation of models. We will give some examples and results, and we will emphasize the importance of relative complexity of processes. Then we will present two classes of application of these results and this prospect.

2. GENERAL METHODOLOGY

A model M is a set of N processes. This model is excited by a sequence of input value V_e (this sequence can be computed by the model it-self). When the simulation is ended, one compute the number of event consumed by each process. One can characterize the performance of a perfect sequential simulator with:

$$T_s = \sum_{i=1}^N T_{ci} \cdot N_i \quad (1)$$

With: T_s is the compute time on a perfect sequential simulator,
 T_{ci} is the compute time for the process i ,
 N_i is the number of events treated by process i

The perfect parallel simulator make use of one processor by process. A process is evaluated every time an event is consumable. A consumable event is an event available at the input in respect with the total order. If a process consumes an event, for this process, there is never an anterior event. The simulation time can be written with the figure :

$$T_r = \text{MAX}_{i=1}^N (T_{ci} \cdot N_i + f_i(M, V_e)) \quad (2)$$

With: T_r is the compute time on a perfect parallel simulator,
 T_{ci} is the compute time for the process i ,
 N_i is the number of event treated by process i
 $f(M, V_e)$ is a function dEpendes on model and input vector d'entrée. this function modelise waitings due to model topology, synchronism, starvation...

Aims of the work is to determine the simulation time for a model by the perfect parallel simulator under the same input sequence than the sequential perfect simulator. When this time is known, it is very easy to compute the natural parallelism of the model in this experience. The natural parallelism is given by:

$$P_n = T_s / T_r \quad (3)$$

If the model is not alterable, there is a non-invasive method, proposed by the author [2], based on the trace of the sequential simulation and a double sequential simulation. In this paper, we suppose that the model is modifiable and specifically models written with VHDL.

We propose below an instrumentation method of VHDL models. This instrumentation allows to extract during a sequential simulation on a non-perfect sequential simulator (time-sharing, time for management of the event- list,...) the same job time on a perfect sequential simulator, the same job time on a perfect parallel simulator and the natural parallelism of the model. The instrumented models in this way are said EAP models. We will observe that results are independent of the instrumentation complexity.

3. ALGORITHM (TWIN CLOCKS)

The instrumentation method and the algorithm is based on the following assumptions :

- one process per processor (no time sharing) for perfect parallel simulation,
- the process i takes, at least, C_i units of time between two consecutive consumptions on the inputs (if events are available, no starvation),
- one process cannot consume an event produced at T_0 time before the time T_0 ,
- In a sequential simulation, events are produced and consumed in total order by each process.

One are going to manage two local (virtual) clocks for each process (these clocks are symbolised by integer variables). These variables represent a time and cannot decrease. The first, T_s , represents the cumulative time of this process and the second, T_r , represents the real time for the processor hosting the process. In addition, each process knows its relative complexity, C (C represents an instruction number, a time,...). Finally, one associate to each event a stamp, $e.H$, which represent the production time of this event. All events which are known at the begin of the simulation (input vector) have a 0 stamp.

During the sequential simulation, when a process consumes an event (in respect with the total order) two cases are possible. Each case induces a behavior of the process :

- 1- $e.H < Tr$: The event is available in the input queue since H, The process can consume it without precaution.
 behavior of process:
 event consumption
 clock update $Tr=Tr+C, Ts=Ts+C$
- 2- $e.H > Tr$: The event will be available in the input queue in the future
 behavior of process:
 event consumption
 clock update in respect with the waiting since
 Tr up to Tr: $Tr=H+C, Ts=Ts+C$

Therefore, the algorithm is :

```

For each process
  initialisation  $Tr=0, Ts=0$ 
  while simulation not finished
    wait for an event
    Consume one event stamped by e.H
     $Tr = \max ( e.H , Tr ) + C$ 
     $Ts = Ts + C$ 
    evaluate outputs
    if outputs are generated, they are stamped by
      s.H = Tr
  
```

One must collect all process clocks to characterize the whole model. At the end of the sequential simulation, or at each step, one must compute the sum of all local Ts and the maximum of all local Tr.

Results are two interesting times Tr and Ts (see part 2). The ratio Tsg/Trg give the natural parallelism of the model.

4. EAP VHDL MODELS

4.1 EAP type: std_logic extension to std_logic_self_e

In order to stamp events, we define a new type `std_logic_self_e` over the IEEE 1164 `std_logic` type.

```

type std_logic_self_e is record
  value:std_logic;
  Tr:integer;
end record;
  
```

These specifications are available through a package `X_IEEE`.

4.2 Global report: resolution functions

In order to compute the relevant times each entity is connected on a global resolved signal. Each entity provides its two internal clocks (Tr and Ts). If one clock of one entity is updated, resolution functions are in charge of computing the sum (`sum_time`) of all Ts and the maximum (`maxi_time`) of all Tr. The resolution functions are built on the scheme :

```

TYPE integer_vector is array (natural range <>) of integer;
FUNCTION maxi_time (V: in integer_vector) return integer;
SUBTYPE rep_time is maxi_time integer;
  
```

```

FUNCTION sum_time (V: in integer_vector) return integer;
subtype sequ_time is sum_time integer;

FUNCTION maxi_time (V: in integer_vector) return integer is
variable max: integer;
begin
    max:=0;
    for i in V'range loop
        if (V(i) >= max) then
            max := V(i) ;
        end if;
    end loop ;
    return max;
end maxi_time;

```

Sum_time function has the same shape. These specifications are available through a package X_IEEE.

4.2 Example of instrumentation of a gate model

With the extended type of logic and resolution functions it is possible to transform a gate level primitive in a EAP model.

```

library IEEE;
use IEEE.std_logic_1164.all;
use WORK.X_IEEE.all;
entity andg is
    generic (tpd_hl:time := 1 ns; tpd_lh:time := 1 ns;
             Tc:integer := 1 ); -- relative complexity
    port (in1, in2 : std_logic_self_e; -- input ports
          out1 : out std_logic_self_e; -- ouput ports
          out_Trr,out_TrS: out integer); -- for global report
end andg;

architecture EAP of andg is
begin
    p1: process(in1.value, in2.value)
        variable val,ex_value : std_logic ;
        variable Trr,TrS,Temp :integer:=0;
        variable out_v:std_logic_self_e:=('U',0);
    begin
        val := in1.value and in2.value;
        Trr:=Trr+Tc; TrS:=TrS+Tc;
        -- detection and clock extraction
        if not(in1.value'quiet) then temp:=in1.Tr; end if;
        if not(in2.value'quiet) then temp:=in2.Tr; end if;
        if (temp > Trr) then
            Trr:=temp+Tc;
        end if;
        -- gate evaluation
        if val /= ex_value then -- no event if same value and new Tr
            ex_value := val; out_v.Tr:=Trr; out_v.value:=val;
            case val is
                when '0' => out1 <= out_v after tpd_hl;
                when '1' => out1 <= out_v after tpd_lh;
                when others => out1 <= out_v;
            end case;
        end if;
    end process;
end architecture EAP;

```

```

        end if;
        -- global report
        out_Trr <= Trr; out_TrS <= TrS;
    end process;
end EAP;

```

All primitives can be transformed in EAP models. A structural description based on these primitives must interconnect all out_Trr and all out_TrS respectively on a max_time and a sum_time resolved signals.

4.3. Instrumentation of synchronous primitive

The previous example show a EAP model for asynchronous system. It is for example). In these example the D flip-flop is modelized at a behavioral level with EAP capabilities.

```

library IEEE;
use IEEE.std_logic_1164.all;
use WORK.X_IEEE.all;
entity DFFC is
    generic (tpd_hl : time := 1 ns;
            tpd_lh : time := 1 ns;
            Tc      :integer := 1 ); -- relative complexity
    port (D,H,C : std_logic_self_e := ('0',0);
          Q      : out std_logic_self_e := ('U',0);
          out_Trr,out_TrS: out integer := 0);
end DFFC;

architecture EAP_BEH of DFFC is
begin
    p1: process(H.value)    -- no sensibility on D input
        variable ex_value : std_logic := 'U';
        variable Trr,TrS :integer:=0;
        variable outv:std_logic_self_e:=('U',0);
    begin
        -- test of the rising edge and updates local clocks
        if (H.Value='1' and H.value'last_value='0') then
            Trr:=Trr+Tc; TrS:=TrS+Tc;
            if (H.Tr > Trr) then
                Trr:= H.Tr + Tc;
            end if;
            outv.Tr:=Trr;    -- update output stamp
            if (C.value='0') then -- synchronous clear
                outv.value := '0';
            else
                outv.value := D.value;
            end if;
            if (ex_value /= outv.value) then -- activity ??
                Q <= outv after (tpd_hl + tpd_lh)/2;
                ex_value:=outv.value;
            end if;
            out_Trr <= Trr; out_TrS <= TrS; -- global report
        end if;
    end process;
end EAP_BEH;

```

4.4 Instrumentation of no reactif system

The two previous examples are reactive systems. They react on their inputs, transform them and product outputs with a new time stamp. It is possible to modelize no reactive systems. The next example shows this property. It is a clock generator model. It has no inputs (no stimuli) and it builds it-self its outputs without dependance on any other part of the design.

```
library IEEE;
use IEEE.std_logic_1164.all;
use WORK.X_IEEE.all;

ENTITY gene_clock IS
    GENERIC (Period: time := 100 ns;
            Tc:integer:=1);    -- relative complexity
    PORT    (H:out std_logic_self_e;
            out_trr,out_TrS: out integer :=0);
END;

ARCHITECTURE classic OF gene_clock IS
BEGIN
    process
        variable sortie:std_logic_self_e:=('0',0);
        begin
            Trr:=Trr+Tc;Trs:=Trs+Tc; -- update clocks
            sortie.Tr:=Trr;    -- update the output time stamp
            sortie.value:=not sortie.value;
            H <= sortie;
            out_Trr <=
                wait for Period/2;
        end process;
END classic;
```

Each event produced by this model has a different time stamp that represent the production of outputs at the maximum throuput.

4.5 . A structural EAP model

A structural description based on EAP models is straightforward. The only extra work required is the interconnection of global reports. The next example is a structural model, based on EAP gates, of a one bit adder.

```
library IEEE;
use IEEE.std_logic_1164.all;
use WORK.X_IEEE.all;

entity adder is
    port (a,b,cin : in std_logic_self_e;
          sum,cout : out std_logic_self_e;
          Trr : out rep_time; Trs : out sequ_time);
end adder;

architecture EAP_STRUCT of adder is
    signal xor1_out, and1_out, and2_out, or1_out : std_logic_self_e;
begin
    xor1: xorg generic map (2 ns,2 ns ,1)
        port map( in1 => a, in2 => b, out1 => xor1_out,
                out_TrS => Trs, out_TrS=>Trs);
```

```

xor2: xorg generic map (2 ns,2 ns ,1)
port map( in1 => xor1_out, in2 => cin, out1 => sum,
out_Trr => Trr, out_TrS=>TrS);
and1: andg generic map (2 ns,2 ns ,1)
port map( in1 => a, in2 => b, out1 => and1_out,
out_Trr => Trr, out_TrS=>TrS);
or1: org generic map (2 ns,2 ns ,1)
port map( in1 => a, in2 => b, out1 => or1_out,
out_Trr => Trr, out_TrS=>TrS);
and2: andg generic map (2 ns,2 ns ,1)
port map( in1 => cin, in2 => or1_out, out1 => and2_out,
out_Trr => Trr, out_TrS=>TrS);
or2: org generic map (2 ns,2 ns ,1)
port map( in1 => and1_out, in2 => and2_out,
out1 => cout, out_Trr => Trr, out_TrS=>TrS);
end EAP_STRUCT;

```

4.6 . Comments

It was shown that it is possible to transform models at any level of abstraction. Moreover, it is possible to build structural EAP models. Results on global Tr and Ts are independent on the complexity of the instrumentation.

5. PROBLEMS AND LIMITATIONS

5.1. Many process in a single architecture

If there are many processes in an architecture, it is difficult to control the local clock for each architecture because there is no shared variable in VHDL (IEEE1076-87). In this case, we create a local signal to allow a global report in the architecture, each process manage its local clocks (each process is associated to a processor).

If there are concurrent statements, they must be transformed in their equivalent process.

5.2 . over-loading of the resolution function

The use of resolution function for global report is very simple but this functions can be over-loaded. If there is many signals in global report, it is interesting to adopt a tree structure for global report.

5.3 . slow down of the sequential simulation

The instrumentation of a VHDL model slows down the sequential simulation. The result is independent on the speed of simulation but if the design is complex before instrumentation the simulation time after instrumentation can become prohibitive.

5.4 . Modification of the model

In order to use the described method, we must modify the model. This modification must be done at behavioral level for local clocks management and at structural level for global report. This modification is not always possible.

6. EXAMPLES AND RESULTS

6.1. Combinatorial system

This example is a structural generic description of a 16 bits ripple carry adder.

Complexity = 96 gates (all gates have relative complexity = 1)

Experiments:

For the elaboration the found Pn is 96.

It is possible to find a test stream with so bad result that Pn=5.6

With random stream on inputs Pn = 39

This example shows that the natural parallelism is very dependent on the input stream, the expected speed-up of the simulation depends on the experiment.

6.2 . Synchronous system

This exemple is a generic synchronous counter with serial report. The model has N flip-flops and N-2 and gates.

It is here very interesting to show the importance of the relative complexity of process (for example the relative execution time). This parameter is never used in the classical method [7][8]

simulation = 5200 clock-edges

N = 16

Number of process = 30

Gate complexity	Flip-flop Complexity	Natural parallelism
1	1	18,46
1	2	17,19
1	5	16,48
1	10	16,24
1	100	16,02
2	1	10,53
5	1	5,78
10	1	4,20
100	1	2,68

6.3 . multi-level stand alone system

The system is a stand-alone structural multi-level description based on:

One N bits Ripple carry adder,

Two N bits synchronous counter,

Two clock generators.

Each clock generator (period 500ns and 1333ns) drives a counter and each counter drives an adder input.

Model Complexity: $10 \cdot N - 2$ processes (158 with 16 bits)

Relative processes complexity = 1

The simulation time is given in 500ns clock ticks.

Elab	100	2898	28,9
6	300	8739	29,13
10	800	9670	12
100	10000	43029	4,3
400	39600	159214	4,02
40000	3999600	15575957	3,89
140000	13999600	54521464	3,89

This result demonstrates that it is possible to instrument mixed level models and to predict the figure $P_n=f(\text{time})$. This figure is often experiment dependent and may be not monotonous.

8. UTILIZATION

8.1. Benchmarking parallel VHDL simulator

With this method it is possible to develop some very well known benchmarks in order to characterize parallel simulator. For example (this design is too simple for actual benchmarking) a parallel simulation on the model of the 6.2 cannot have a speed up greater than P_n .

If this parallel simulation provides a speed-up $S=5$, we can accurately say that the simulator extracts S/P_n of parallelism. (In this example the extracted parallelism is 27% (relative complexity of (gate, flip-flop) is (1,1)). On the other hand if the (gate, flip-flop) complexities are (5,1) the extracted parallelism is 86.5%. A very rough result, sometimes written, characterizes the parallel simulator by the ratio S/N , 16,6% in this example, without taking the P_n limit and relative process complexities into account.

8.2. Approaching the optimal number of processor

We think that the natural parallelism is a measure close to the optimum processor number for a parallel simulation. If a model has a P_n of 12, it is silly to try a simulation on a hundred processors machine even if the model has hundreds processes.

We work now on the characterization of the optimal target machine based on the natural parallelism and extended dependency graph of models.

9. OUTLOOKS

9.1. Data-base of benchmarks

Based on EAP models, we are working on the design of a benchmark set which represents a very large class of applications and complexity (from few gates to more than thousands, with combinatorial, synchronous, hybrid designs).

The data-base constitution is in progress by instrumentation of models provided by P.ASHENDEN (Adelaïde, Australia), T.Collette (CEA/France), F.PEcheux (MASI/France), M.Markowitz (EDN asia editor)....

9.2. Enhance and automate the method

In order to automate the instrumentation, we think about an encapsulation method of each process in a shell that manages local clock and output time stamping.

9.3. VHDL Virtual processor

It is very useful to know the natural parallelism of a model, but the aim is to distribute this model on processors of a actual machine in order to reach the greatest speed-up. We work on a characterization method of the extracted parallelism of a parallelization of a N processes model on a M processors machine ($M < N$)

This method is based on the VHDL model of a perfect processor which manages some VHDL processes (the model to simulate). The clock updates, and the process scheduling will be emulated by this virtual processor. The correctness of the result is due to the use of a sequential simulator that ensure the total order of events.

The expected result is the maximum extracted parallelism of a processes distribution of a VHDL model on a parallel machine.

10. CONCLUSION

In this paper, we have presented a meta-utilization of the VHDL language. With VHDL constructs and algorithms, we can characterize a VHDL model, more exactly the VHDL model characterizes itself through a sequential simulation in order to extract the natural parallelism of the model.

The interest of the method is its simplicity and the use of a sequential simulator to obtain results on parallel simulation.

Results can be used to characterize the quality of a distributed simulator or to evaluate the number of processor in the cluster which will run the simulation with good performances.

BIBLIOGRAPHY

- [1] T.COLLETTE
Architecture et validation comportementale en VHDL d'un calculateur parallèle dédié à la vision par ordinateur
INP Grenoble Thesis, Microelectronique (14 sept 92)
- [2] Y.HERVE
Accélération maximum théorique d'une simulation répartie à partir de la trace d'une simulation séquentielle.
Submit to Technique et Science Informatique (TSI) (november 1992)
- [3] P.INGELS, M.REYNAL
Simulation répartie: schémas d'exécution pour un modèle à processus
TSI Vol 5, no 9, 1990, pp 383-397
- [4] N.ISHIURA et al.
Time first evaluation algorithm for high speed logic simulation
ICCAD-84,IEEE Novembre 1984, pp 197-199
- [5] D.R.JEFFERSON
Virtual Time. ACM transactions on programming Languages and Systems
Vol 7, no 3, july 1985, pp 404-425
- [6] J.MISRA
Distributed Discrete-Event Simulation
Computing Surveys, Vol 18, no 1, March 1986, pp 39-65
- [7] L.SOULE, A.GUPTA
Parallel Distributed-Time Logic Simulation
IEEE Design & Test of Computers, pp 32-48, DEcembre 1989
- [8] L.SOULE, T.BLANK
Statistics for parallelism and abstraction level in digital simulation
DAC 1987, pp 588-591