

A Parallel VHDL Simulator on the Connection Machine

Alan D. Cabrera
Computer Laboratory
Michigan State University
East Lansing, Michigan 48824-1042
adc@tardis.cl.msu.edu

Prof. Moon Jung Chung
& Yunmo Chung
Computer Science Department
Michigan State University
East Lansing, MI 48824
chung@cps.cps.msu.edu

Abstract

This paper proposes a strategy to implement a parallel VHDL simulator for fast simulation on the Connection Machine. The VHDL description for a circuit is translated into an intermediate form for simulation. The intermediate form is simulated with discrete event simulation techniques, such as Time Warp and the Chandy-Misra algorithm, on the Connection Machine. Signal assignment statements in the VHDL description are transformed into gate processors, and component instantiations are handled by their expansion into a combination of basic elements. Experimental results of VHDL simulators with several simulation techniques are given. Compared with the VHDL simulator on the Sun 3/280, the parallel simulator using the Chandy-Misra algorithm on the Connection Machine is much faster by several orders of magnitude.

Categories

1.2 Discrete Simulation
5.1 Behavioral and Hardware Description Languages

1 Introduction

In the behavioral simulation of digital circuits, the behavior of a circuit is described using a hardware description language. The VHDL is a standard hardware description language [9]. As one of ways of achieving fast simulation, we present a strategy for a parallel VHDL simulator that runs on the Connection Machine. Our parallel VHDL simulator consists of two parts, VHDL translator and parallel simulator. The translator translates a VHDL description into a predefined intermediate form. The parallel simulator uses the intermediate form as its input. The simulator can be implemented based on one of distributed discrete simulation techniques on the Connection Machine. While VHDL programs include both behavioral and structural descriptions, our initial translator, because of constraints imposed by our simulator, accepts a subset of the VHDL structural descriptions. This subset includes the simulation of inertial delays and guarded statements.

There are two general simulation approaches for distributed discrete event simulation [14]: *synchronous* and *asynchronous*. In synchronous event simulation, there is a global event queue which contains all the events sorted by their timestamps. The events with smallest timestamps are chosen and executed in parallel. In asynchronous parallel simulation, each process has its own input message queue(s) and local clock. More than one event can be executed simultaneously without concerning about event processing precedence. A process executes arriving events according to a certain scheduling policy.

Again, asynchronous simulation can be divided into two common simulation paradigms: *conservative* and *optimistic*. The Chandy-Misra algorithms [3] are the most well-known *conservative* paradigms. A logical process will block until it has been confirmed that no other logical processes' simulation time is less than its own. This kind of blocking can lead to deadlock and requires that simulations run in parallel with deadlock breaking mechanisms. The Time Warp mechanism, proposed by Jefferson [10], [11], [12], is the most popular *optimistic* paradigm. Logical processes execute events asynchronously, not blocking to wait for other logical processes to "catch up," until conflicting information appears; i.e. event precedence has been violated. When this happens a *rollback* must occur, reverting back to a previous state that existed before the event precedence viola-

tion. Hence in Time Warp there is a certain amount of overhead involved with the storage and management needed for rollback.

As a simulator for parallel simulation, several approaches have been proposed, such as data parallel Time Warp [4] and the Chandy-Misra algorithm [5]. We used the data parallel Time Warp approach to implement the parallel VHDL simulator. Using the Connection Machine's massive parallelism, execution times for simulations of structural descriptions are drastically reduced.

In Section 2 "Preliminaries" we describe the data parallel implementation of Time Warp on the Connection Machine. In Section 3 "Characteristics of VHDL for Parallel Simulation" several characteristics in VHDL are examined for translating VHDL descriptions into the input form for a given parallel simulator. Section 4 "Translation of VHDL Descriptions" describes how certain VHDL descriptions are translated into the intermediary form that the simulator accepts. Implementation experience and performance evaluation are given in Section 5 "Implementation Experience and Performance Evaluation". Conclusions are presented in Section 6 "Conclusions".

2 Preliminaries

2.1 The Connection Machine

The Connection Machine [19] consists of 16K to 64K processors, interconnected on both a hypercube network and two dimensional grid. Each processor has 8KB of memory. The method of control is Single Instruction Multiple Data stream (SIMD). Processors can be selected by on either the type of data stored in the processors, or by any boolean expression on that processor. Several operations can be implemented efficiently on connection machine, including "parallel prefix computations" and "packing" [8].

Research on the simulation of circuits on the Connection Machine have been reported with results that have limited applications, or the simulation model is of a lower, circuit, level [2], [21]. Chung and Chung proposed parallel logic simulation techniques [4], [5], on the Connection Machine using discrete event simulation. Our simulator uses a data parallel simulation technique [4] among them.

In the Connection Machine, several virtual processors can be assigned to a single physical processor on the machine, thus providing more processors when needed than that are physically available.

2.2 Data Parallel Logic Simulation

In [4], each circuit element and each event are dynamically assigned to a processor (called *gate* processor, and *event* processor, respectively) on the Connection Machine. In addition, input waveforms represented by input events are allocated event processors before event simulation starts. A gate processor contains information about the corresponding gate circuit, i.e., delay, function, successors, etc. An event processor contains its event-simulation time, event signal value, pointer to the gate processor to execute the events, etc. Gate processors interact by sending messages to one another, each one time stamped with its simulation time. Event processors which have pointers to the same gate processors are grouped together in their own segment. This facilitates the efficient computation of the Local Virtual Time (LVT). Because of the hypercube based Connection Machine architecture, the LVT of all gates can be computed simultaneously using "segmented-min". The Global Virtual Time GVT can be computed using "global-min" operation.

A scheme in which each object has a single event queue instead of three queues needed in the conventional Time Warp protocol, is developed for fast queue manipulation and small storage requirement. To minimize the number of events at a process over the simulation, a lower bound of rollback (LBR) [4] at each process is computed and used to discard processed events with timestamps less than the bound. To reduce rollback frequency and space overhead of the Time Warp protocol, Moving Time Window (MTW) [16] is used. A global window is given to limit the maximum difference between the local virtual simulation time of communicating objects.

3 Characteristics of VHDL for Parallel Simulation

In this section we discuss our implementation of a VHDL simulator using the Time Warp paradigm. Modifications to the above techniques are discussed and the interfaces between the translator and simulator are elaborated.

A subset of VHDL which includes all concurrent statements is selected, including block statements with guard expressions, conditional signal assignments, and component instantiation statements. Both of the two different delay models, inertial and transport delays are also included in the subset. Initially, we include only the type bit.

In a behavioral simulation on SIMD, we cannot issue different instructions at the same time to simulate different types of circuit elements. That is, if the behavior of two circuit elements are different, the instructions which simulate the circuit elements are different so that we cannot simulate them simultaneously. To handle this problem, the VHDL description of the circuit is broken into smaller circuit elements so that the function of each circuit element can be computed by table look-up from its inputs. A signal assignment statement is transformed into a combination of logic gates with delay. To simulate *guarded statements*, a *guard gate*, is introduced. Using *guard gates*, we can also simulate conditional signal assignments. If there is a component instantiation in the VHDL description, the component is expanded into a combination of basic elements by the translator.

Currently our simulator takes a simulation specification from the VHDL translator and simulates the circuitry using Time Warp on the Connection Machine. The VHDL translator performs syntactic and semantic checking and creates the simulation specification. Each signal assignment statement is translated into a gate or event processor. Additional gate and guard processors, may be required if the statement is guarded. The translator supplies information about the gate needed by the simulator such as target, delay type and length, function to be simulated, input signals, etc. The translator also supplies information about event processors: simulation time, event signal, target, etc.

3.1 Processor Memory Layouts

We consider the memory layouts of gate and event processors. The layout of the guard processor is similar to that of the gate processor and is described in Section 3.3 "Simulation of Guarded Statements". The memory layout of a gate processor, shown in Figure 1: "The memory layout of the gate processor", contains the following fields:

1. **target:** The target of the gate processor.
2. **delay type:** The type of delay that the gate should simulate, inertial or transport.
3. **delay:** The length of the delay for the gate. Should the length of the delay be 0, the length actually used is 1ps.
4. **function:** The operation that the gate performs. The function of the gate is implemented by a table created by the translator. If maximum number of input gates for a function table is more than four, then it's corresponding gate is broken up into multiple gates such that each function table has no more than four inputs. This new set of gates will simulate the semantics of the old gate they replace.
5. **input signals:** The inputs to the gate.
6. **bound events:** The upper bound event processor numbers of segments allocated for the previous gate and the gate respectively.
7. **active bit:** The bit indicating whether the gate is active or not.
8. **LVT:** The Local Virtual Time of the gate, initially set to -1.

Figure 1: The memory layout of the gate processor

target
delay type
delay
function
input signals
bound events
active bit
LVT

Figure 2: The memory layout of the event processor

simulation time
tag
event signal
input signals
in-pin name
ptr to gate processor

The layout of memory for the event processor, shown in Figure 2: "The memory layout of the event processor", is thus:

1. **simulation time:** The simulation time when this event will be executed.
2. **tag:** The tag indicates whether the event has been executed or not.
3. **event signal:** The signal value generated by the computation of the previous gate.
4. **input signals:** The input signal values at the gate just before the event signal is generated. This is used in case of a rollback, we will use the input signals to restore the input signals field of the gate.
5. **in-pin name:** The name of the pin position of the gate on which the event will be executed.
6. **pointer to gate processor:** The pointer at which the event will be executed.

3.2 Simulation of Inertial Delays

A VHDL simulation has two delay models for modeling switching circuits [9], *transport* and *inertial*. Transport delay is a characteristic of hardware devices where a pulse is transmitted no matter how long its duration is. Inertial delay is a characteristic of switching circuits where a pulse whose duration is shorter than the switching time of the circuit will not be transmitted. Most logic simulators are based on using either the unit delay or the transport delay model [4], [13], [17], [21]. In this subsection we present the additions and modifications to the simulator, described in Section 2 "Preliminaries", required to support the simulation of inertial delays.

For inertial delay, should a pulse's duration be shorter than the switching time of the circuit, the trailing edge of the pulse must annihilate the leading edge of the pulse and cause a roll-back; hence the pulse will have virtually not been transmitted.

Let t be the time when the pulse initially changes for a circuit, g , that has an inertial delay. If the inertial delay of g is Δ_g , then an event message will be sent with a time stamp $t + \Delta_g$. When the state of the pulse changes, say at time t' , two events will be created. First an event message with a time stamp of $t' + \Delta_g$ will be delivered. Second an anti-event message will be delivered. Its scope of "destruction" will be only those events that share the same origin and have a time stamp $t | t' - \Delta_g < t < t'$. Because anti-events are used to simulate inertial delay, *garbage collection* needs to be carefully implemented.

Garbage collection discards events which will never be used for further rollback procedures again, and to compact all remaining events. The scheme in [4] was to discard all those events whose simulation time were smaller than the GVT. Care must be taken in the garbage collection such that anti-events can, if needed, properly annihilate their event message counterparts.

3.3 Simulation of Guarded Statements

The simulation of guarded concurrent signal assignment statements in VHDL requires the addition of a guard processor. Consider the semantics of guarded signal assignment using Figure 4: "Guarded signal assignment." For block *b* the guard expression is `clk = '1'`. The signal assignment statements are guarded statements. When `clk = '1'` is true, changes in the waveforms are allowed to propagate to the targets *s1* and *s2*. When `clk = '1'` is false, no changes are propagated to the targets and the values of the targets are their previous values before `clk = '1'` became false. When `clk = '1'` becomes true again, the values of *s1* and *s2* will be updated to reflect the current value of their respective waveforms. Figure 3: "Waveforms of guarded signal assignment statement" illustrates the waveforms of the above example.

Figure 3: Waveforms of guarded signal assignment statement

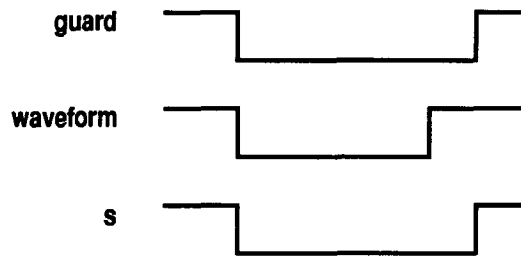


Figure 4: Guarded signal assignment.

```
b: block (clk = '1')
begin
    s1 <= guarded w1;
    s2 <= guarded w2;
end block;
```

(a) VHDL description

To simulate the semantics of guarded statements we introduce the guard processor and its gate processor. The gate processor is used to evaluate the guard expression. When the guard expression is true (false) a '1' ('0') is sent to the guard processor. The advantage of this method is that one gate processor is needed for the entire set of guarded processors in a block. Should the function table of the guard expression need to be broken down, only the gate processor is affected, as opposed to the entire set of guard processors.

The guard processor will inhibit the transmission of waveform signals to the target when its input from the guard expression is false. When the guard expression returns to true, the guard processor will update the target to the current value of the waveform by sending an event message.

The memory layout for the guard processor is shown in Figure 5: "The memory layout of the guard processor". This layout is similar to that of the gate processor. The guard processor takes its input from the waveform gate processor and the guard expression gate processor. The function field of the guard processor is just the identity function, hence it is left empty.

Figure 5: The memory layout of the guard processor

target
delay type
delay
.
input signals
bound events
active bit
LVT

4 Translation of VHDL Descriptions

The translator translates VHDL structural descriptions into the intermediate form of the simulation specification required by the simulator. Our translator is written in C++ [18], BISON¹, and LEX, and uses Gorlen's NIH class Library [7]. We show how various constructs of a VHDL description are translated into processor descriptions needed by the Time Warp simulator. The translated simulation specification will be illustrated by using the memory layout of the gate, guard, and event processors. Figure 6: "Different views of a signal assignment statement" compares the actual output of the translator, translating a signal assignment statement, with the memory layout of the allocated processors. The simulator uses the output from the translator and allocates processors and their memory.

Additional work is performed by the translator when components are used in a VHDL description. Care must be taken such that internal variables of component instantiations of the same design entity do not pose a problem, i.e. scoping rules defined by [9] are enforced. Section 4.2 "Sample Translation" gives an example as to how component instantiations are translated.

The sequence of translation resembles that of elaboration. The top design entity is translated, by recursively translating its nested constructs, into the simulation specification. All of the blocks in the design entity are translated by translating their signal assignment and component instantiation statements. The translation of signal assignment statements in these blocks are described in Section 4.1 "Concurrent Signal Assignment". The translation of component instantiation statements in the blocks are handled by proper association of formal parameters with actual parameters and a recursive translation of the design entity specifying the design of the instantiated component.

-
1. BISON is a YACC-like utility [6].

Figure 6: Different views of a signal assignment statement

```
s <= clr = '1' and hold after 2 ns;
```

(a) Statement for translation.

```
Gate[
  Name[gate@7]
  Target[s]
  Inertial[2ns]
  Function[
    0
    0
    0
    1
  ]
  Input[
    clr
    hold
  ]
]
```

(b) Simulation specification from translator. Note that the function table is represented by a list of boolean values. The values 0, 0, 0, and 1 in the field **Function** denote the function values of the gate when the values of (clr, hold) are (0,0), (0,1), (1,0), and (1,1), respectively.

gate	
s	
Inertial	
2ns	
clr = '1' and hold	
clr hold	
-	
-	
-	

(c) The memory layout of the allocated gate processor.

4.1 Concurrent Signal Assignment

Concurrent signal assignment statements are translated into a number of gate and guard processors, each translation depending upon the type of signal assignment performed, *conditional* or *selected*. In this section we elaborate the translation of conditionals signal assignment. The translation of selected signal assignment can be done similarly.

Conditional signal assignment statements are translated into a number of gate and guard processors. Each waveform element of a waveform is translated into a gate processor. An additional guard processor is added if the waveform is a part of a when-else clause. The function field of the guard is a logical expression created by the translator to ensure the correct "if-then-else" semantics of the when-else clause. Should the statement be guarded, an additional guard and gate processor will be allocated to guard the target. A block illustration can be seen in Figure 7: "Guarded condition signal assignment." on page 9.

Figure 7: Guarded conditional signal assignment. Here "G" denotes a gate processor, "g1" is the gate for the guard expression, "s" is the target, "e1" is the gate for the expression of the conditional signal assignment statement, and "w1" and "w2" are the gates for the waveform expressions.

```

b: block (g1)
  begin
    s1 <= guarded w1 when e1 else
          w2;
  end block;

```

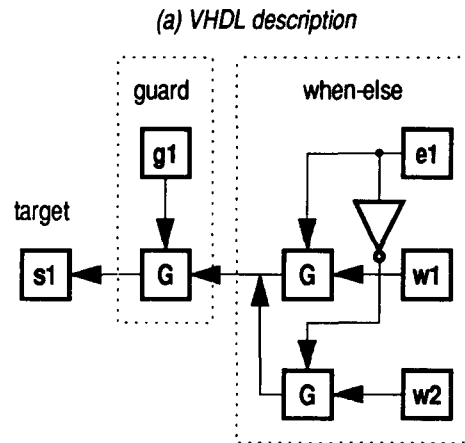


Figure 8: "Guarded conditional signal assignment" shows the translation of the description in Figure 9: "Translation of conditional signal assignment" .

Figure 8: Guarded conditional signal assignment

```

entity dff is
  port (clk, clr, c, d: in bit;
        s: out bit);
end dff;

architecture ci_dff of dff is
  begin
    b: block (clk = '1')
      begin
        s <= guarded c after 1 ns when clr = '1' else
              d after 2ns;
      end block
    end ci_dff;

```

4.2 Sample Translation

Listed in Figure 11: "Sample VHDL description" is a sample VHDL description. This example is a structural description of a two-bit adder using instantiations of full one-bit adders. These instantiations are inside the architecture definition `w_structure` of the entity `two_bit_adder` and are labeled 1a and 1b. Figure 10: "Basic component connections" illustrates the overall connections between the components.

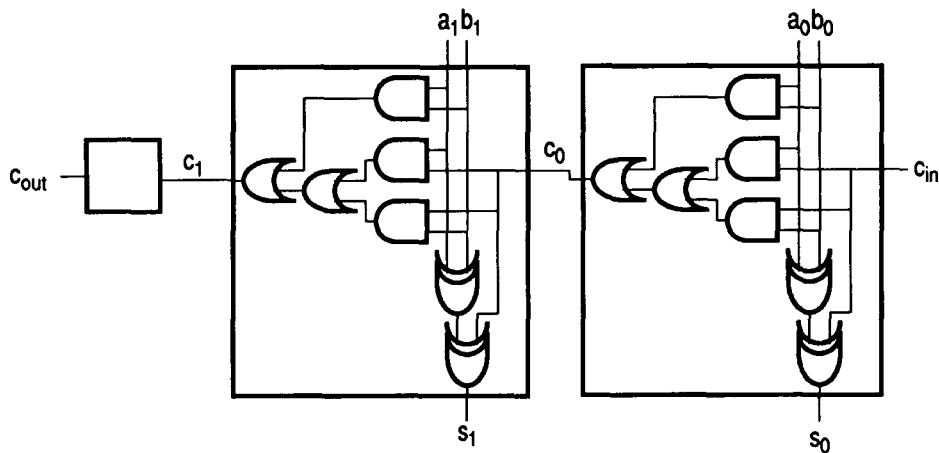
Notice the signal `s` inside the architectural definition `data_flow`. While it does not contribute to the semantics of the adder, it has been added to illustrate that proper scoping of signals when component instantiation is performed.

Figure 12: "Translated VHDL description" on page 13 contains the intermediate output of the VHDL translator.

Figure 9: Translation of conditional signal assignment gate

gate	gate	gate	gate	gate
n@1	n@2	n@3	s	n@3
Inertial	Inertial	transport	Inertial	transport
1ns	2ns	1ps	1ps	1ps
c	d	clr = '1'	clr = '1'	not (clr = '1')
c	d	n@1	n@3	n@2
.
.
.

Figure 10: Basic component connections



5 Implementation Experience and Performance Evaluation

In this paper we presented a method of translating VHDL descriptions into simulation specifications for the Connection Machine. Even though the Connection Machine is a synchronous SIMD machine, the VHDL descriptions of the circuit can be broken into basic gate elements such that they can be simulated asynchronously using Time Warp.

The implementation of the translator was made easier using object oriented design and programming techniques. The programming effort was greatly simplified by using the NIH class library.

Using the techniques of software reusability, it should be relatively straight forward to reuse software components used in the initial translator for new translators for simulators that use a paradigm other than Time Warp.

Figure 11: Sample VHDL description

```

entity full_adder is
  port (x, y: in bit;
        cin: in bit;
        cout, sum: out bit);
end full_adder;

architecture data_flow of full_adder is
  begin
    main: block
      signal s: bit; -- internal state
      begin s <= x and y; -- a contrived signal assignment
            sum <= x xor y xor cin after 1 ns;
            cout <= (y and cin) or (x and cin) or s after 2 ns;
      end block;
    end data_flow;

entity two_bit_adder is
  port (a1, a0, b1, b0: in bit;
        s1, s0, cout: out bit);
end two_bit_adder;

architecture w_structure of two_bit_adder is
  begin
    w_structure_blk: block
      signal cin: bit;
      signal c0, c1: bit;
      component fadder
        port (x, y, cin: in bit; cout, sum: out bit);
      end component;
      for all: fadder use entity full_adder(data_flow);

      begin
        la: fadder port(x=>a0, y=>b0, cin=>cin, cout=>c0, sum=>s0);
        lb: fadder port(x=>a1, y=>b1, cin=>c0, cout=>c1, sum=>s1);
        cout <= c1 after 7 ns;
      end block;
    end w_structure;

```

Difficulties were encountered in the design of the translator because VHDL descriptions had to be translated into the gate level descriptions used by the simulator. This limited the initial implementation of the translator to only handle structural VHDL descriptions. Other aspects of VHDL simulation had to be considered such as simulation of inertial delay and guarded statements

Table 1: "Execution time (in seconds) for a 32-bit array multiplier" shows the performance of the parallel Time Warp approach on the COnnection Machine and the VHDL simulator on Sun 3/280. The data parallel Time Warp simulation for a 32-bit array multiplier with 8256 gates is much faster than the VHDL simulation. The performance evaluation for bench mark circuit ISCAS/C6288 [1] with 2416 gates is given in Table 1: "Execution time (in seconds) for a 32-bit array multiplier".

Table 1: Execution time (in seconds) for a 32-bit array multiplier

# of Input vectors	1	2	4	6	9
VHDL	111	203	420	700	1023
data parallel time warp	2.0	3.8	5.1	8.0	10.2

The parallel simulator is much faster than the VHDL simulator on the sequential machine.

Compiled mode simulation techniques have been reported to give good performance for logic simulation [15], [16]. However, the techniques cannot be used for parallel VHDL simulation because of the following reasons. First, the compiled mode simulation technique is only good for unit-delay simulation. Second, in general, it lacks the ability to handle asynchronous circuits. Therefore, the technique limits its application to combinational and synchronous circuits.

Table 2: Execution time (in seconds) for bench mark circuit ISCAS/C6288

# of Input vectors	10	20	30	40
data parallel time warp	4.5	8.2	12.1	15.4

In this paper, we focused on the VHDL simulation based on the data parallel Time Warp simulation with the Connection Machine. With the lower bound of rollback (LBR) computation and the Moving Time Window (MTW) mentioned in Section 2, we could achieve the significant reduction of rollback frequency and space overhead and thus good speedup in the Time Warp implementation. Several other parallel logic simulation approaches have been reported in [5]. Note our implementation can be extended to the other techniques.

6 Conclusions

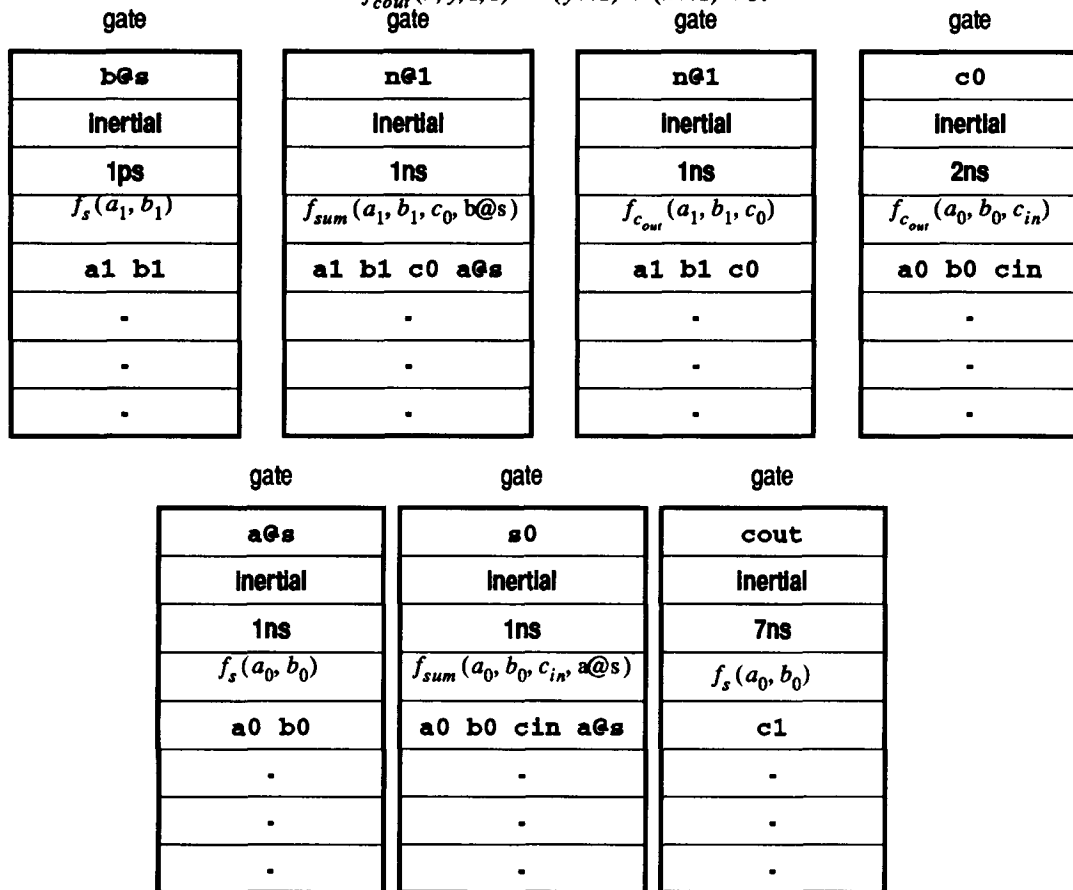
Our implementation of a VHDL simulator uses data parallel Time Warp techniques and the Connection Machine's massive parallelism for performance improvements of several orders of magnitude. Once the initial difficulty of the translator specification was completed, due to constraints placed by the simulator, implementation of the translator was relatively easy using object oriented design and programming techniques.

For further research, we plan to include behavioral descriptions and extend the set of available data types. We also plan to evaluate the use of other paradigms available for parallel VHDL simulation such as process-oriented Time Warp and Chandy-Misra. Note that our simulation scheme, such as decomposing the circuits into basic gate elements and run the simulation specification on the parallel machine, can also be applied to other types of parallel machines such as the BBN Butterfly, a shared memory MIMD machine. Hence performance on other machine architectures will also be explored.

References

- [1] F. Brglez, P. Pownall, and R. Hum. Accelerated ATPG and fault gradient via testability analysis. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, June 1985.
- [2] R.E. Bryant. Data parallel switch-level simulation. In *Proceedings of the 1988 International Conference on Computer Aided Design*, 1988.
- [3] K.M. Chandy and J. Misra. Asynchronous distributed simulation via a sequence of parallel computations. *Communications of ACM*, 24(11):198-206, April 1981.
- [4] M.J. Chung and Y. Chung. Data parallel simulation using time-warp on the connection machine. In *Proceedings of the 26th Design Automation Conference*, pages 98-103, June 1989.
- [5] M.J. Chung and Y. Chung. Efficient parallel logic simulation techniques for the connection machine. In *Proceedings of the Supercomputing '90*, November 1990.
- [6] C. Donnelly and R. Stallman. *BISON*, Free Software Foundation, October 1988.
- [7] K.E. Gorlen, S. M. Orlow, and P. S. Plexico. *Data Abstraction and Object-Oriented Programming in C++*. John Wiley and Sons, 1990.
- [8] W. D. Hillis. *The Connection Machine*. MIT Press, 1988.
- [9] Institute of Electrical and Electronics Engineers, Inc., New York, NY. *IEEE Standard VHDL Language Reference Manual*, IEEE Std 1076-1987 edition, 1988.
- [10] D. Jefferson. Implementation of time warp on the caltech hypercube. *Society for Computer Simulation Multiconference*, pages 63-69, January 1985.
- [11] D. Jefferson. Virtual time. *ACM Trans. Programming Languages and Systems*, 7(3):404-425, July 1985.

Figure 12: Translated VHDL description where $f_s(x, y) = x \wedge y$, $f_{sum}(x, y, z) = x \text{ xor } y \text{ xor } z$,
 $f_{cout}(x, y, z, s) = (y \wedge z) \vee (x \wedge z) \vee s$.



[12] D. Jefferson and H. Sowizral. Fast concurrent simulation using the time warp mechanism. *Society for Computer Simulation Multiconference*, pages 63-69 January 1985.

[13] J. V. Briner Jr., J. L. Ellis, and G. Kedem. Taking advantage of optimal on-chip parallelism for parallel discrete event simulation. In *Proceedings of the IEEE International Conference on Computer-Aided Design*, pages 312-315, 1988.

[14] Y. Lin, E.D. Lazowaska, and M.L. Bailey. Comparing synchronization protocols for parallel logic-level simulation. In *Proceedings of the 1990 International Conference on Parallel Processing*, volume 3, pages = 223-227, 1990.

[15] P.M. Mauer and Z. Wang. Techniques for unit-delay compiled simulation. In *Proceedings of the 27th Design Automation Conference*, pages 480-484, 1990.

[16] L.M. Sokol, B.K. Stucky and V.S. Hwang. MTW: A control mechanism for parallel discrete simulation. In *Proceedings of the 1989 International Conference on Parallel Processing*, volume 3, pages 250-254, 1990.

[17] L. Soul and T. Blank. Parallel logic simulation on general purpose machines. In *Proceedings of the 25th ACM/IEEE Design/IEEE Design Automation Conference*, pages 166-171, 1988.

[18] B. Stroustrup. *The C++ Programming Language*. Addison-Wesley, 1986.

[19] Thinking Machines Corporation. *Using the Connection Machine*. May 1988.

[20] Z. Wang and P.M. Mauer. LECSIM: A leveled event driven compiled logic simulator. In *Proceedings of the 27th Design Automation Conference*, pages 491-496, 1990.

[21] D. M. Webber and A. Sanggiovanni-Vincentelli. Circuit simulation on the connection machine. In the *Proceedings of the 24th ACM/IEEE Design/IEEE Design Automation Conference*, pages 108-113, 1987.