Temporal Parallel Gate-level Timing Simulation

Dusung Kim, Maciej Ciesielski
University of Massachusetts
Dept. of Electrical & Computer Engineering
Amherst, MA 01003, USA
{dukim, ciesiel}@ecs.umass.edu

Kyuho Shim, Seiyang Yang
Pusan National University
Department of Computer Engineering
Busan, Korea
{capnemo, syyang}@pusan.ac.kr

Abstract – *This paper introduces a radically different approach to parallel simulation for gate level design, aimed at completely eliminating the communication and synchronization overhead between simulators. It is based on a new concept of temporal parallel simulation: in contrast to traditional, spatially-distributed simulation, which partitions the design into multiple modules to be simulated concurrently, the proposed temporal parallel simulation partitions the single simulation run into multiple simulation runs in temporal domain. Experimental results demonstrate that linear speedup is possible for large designs and long simulation runs.*

1. Introduction

Unlike other verification methods (formal verification, or hardware assisted simulation), it provides full signal visibility and scales with design size. Simulation is also being used to complement static methods, such as static timing analysis (STA) and formal verification (equivalence checking) for timing verification [1].

However, the major drawback of simulation is its low speed. There have been several approaches to address this deficiency, such as hardware-assisted simulation acceleration [2], distributed parallel simulation [3] and abstraction level design simulation [4].

This paper concentrates on a distributed approach to simulation, as a means to parallelize the simulation and improve its overall performance. A new approach to parallel HDL simulation is proposed, based on a concept of *temporal* parallel simulation, rather than traditional, *spatially* distributed simulation. Unlike the spatial simulation method, the method described in this paper does not introduce any dependency among the simulation nodes. The synchronization and communication overhead due to inter-module communication and signal passing, characteristic of the traditional distributed simulation is *completely* eliminated. This is a very important feature, which increases parallelism and maximizes speedup ratio with respect to the number of simulation nodes.

In this paper we outline the concept of temporal parallel simulation and show that it is compatible with the accepted design and verification flows. We discuss technical issues to make this method practical and show some preliminary experimental results.

The proposed technique is universally applicable to full timing simulation of designs of any size and type (both with single and multiple, asynchronous clocks). However, for the purpose of this paper we confine its description to gate-level designs with a single clock. It is worth mentioning that gate-level timing simulation is still being used, even for designs with a single clock, because STA alone is not efficient for timing verification due to gated clocks, false paths, multi-cycle paths, etc [5].

2. State of the Art

A rich body of literature exists in the area of parallel simulation; however most of the known work addresses traditional parallel simulation, which is based on physical *partitioning of the design* into modules, distributed to individual simulators. We refer to this approach as *spatial parallelism*, since the simulation relies on partitioning of the design in spatial domain. This form of parallel simulation has been known since late 1980s as *Parallel Discrete Event Simulation* (PDES) [6],[7],[8],[9],[10]. Such an approach to distributed simulation suffers from an unavoidable

communization and signal synchronization overhead between the modules, and a lack of methods to perform efficient design partitioning to minimize this overhead. To the best of our knowledge, most of the current research in distributed HDL simulation is based on this approach. The results have been demonstrated on relatively small or well structured designs that can be easily partitioned without incurring large communication cost.

Li *et al*. [11] implemented a parallel Verilog simulator by adopting an Object-Oriented concept. However, its performance is still not acceptable because of the high synchronization and communication overheads for real, complex designs.

Zhu *et al*. [12] attempted to apply this approach to realistic designs and achieved good event processing rate with the number of events growing linearly in the number of processors. In spite of these promising results, the actual simulation performance can easily deteriorate, as it strongly depends on the design structure and the dynamic behavior of the design.

No clean solution has been proposed to address the traditional issues plaguing this approach, such as synchronization, message passing/sharing, and design partitioning problems. As a result, only a few commercial products have been developed for such spatial parallel simulation, including SimCluster [13] and MP-Sim [14]. To this date they, however, were not able to attract attention of expert designers.

3. Temporal Parallel Simulation

The temporal parallel simulation proposed in this paper is a radical departure from the conventional parallel HDL simulation for gate level timing simulation. In contrast to spatial parallelism, which partitions the design to be simulated, our temporal parallel simulation partitions the simulation run in time, by cutting the entire simulation run into a number of independent simulation slices. It consists of two major steps:

1. Fast, high-speed, high-level reference simulation on a single processor that stores essential information at selected checkpoints; and
2. Low-level target simulation, which is distributed to the individual processors.

The reference simulation runs at zero-delay gate level (GL) of design abstraction, while the target simulation is run at the full-timing GL, using annotated SDF (Standard Delay Format). Usually full-timing gate level simulation is 10~50 times faster than full-timing gate level simulation. This large difference in simulation speed makes it possible for the proposed approach to achieve significant speedup improvement. The basic concept of this new technique is shown in Figure 1.

Both steps of the simulation work on the entire design under test (DUT), while the entire simulation run is divided into simulation *slices*, each to be executed on an independent simulator.

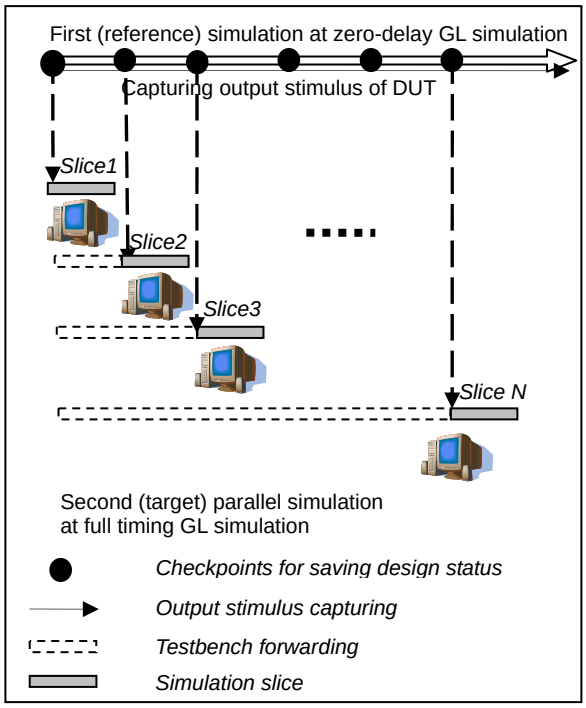

Fig. 1. Concept of temporal parallel simulation.

For this approach to work, the initial design state for each slice of the target simulation must be captured and saved during the first (reference) run.

This is done at predetermined *checkpoints*, determined by the number of processors available for distributed parallel simulation. The design state consists of the state of all internal registers and memory print of the design. By restoring the design states, each slice can be made independent of each other. As a result, target simulation can run concurrently and independently for each slice.

3.1 Testbench Forwarding

While the design state of DUT can be stored at any point during the simulation, the state of the testbench cannot be similarly captured. This is because the state of the testbench (software) is not clearly defined. In order to maintain the correct testbench state at the restoring point, the testbench must be simulated from the beginning of the simulation time to the initial time of each simulation slice. This *testbench forwarding* technique is a fast, testbench-only simulation used in our temporal parallel simulation.

During the reference simulation, the values of all output ports and bi-directional ports of DUT are captured. The captured data is then used to create a *dummy* DUT, which has exactly the same output behavior as an actual DUT under the zero-delay gate-level model. Such constructed dummy DUT is used as a stimulus provider for the simulation of the testbench up to each restoring point prior to target simulation of each slice. After restoring the design state, the dummy DUT is replaced by the original DUT. Note that the testbench is usually implemented in RTL or higher abstraction level. Moreover, because the complex simulation-time intensive DUT is completely eliminated during this phase, the testbench forwarding technique effectively reduces the testbench simulation overhead.

3.2 Maintaining Functionality of Testbench

An important feature of our approach is that it offers a significant performance improvement of simulation without sacrificing the verification capability of the testbench. Throughout the simulation, no modification to the original testbench is made. Therefore, all the verification features defined in the testbench are applicable to temporal parallel simulation.

4. Correctness of Simulation

The correctness of our approach can be verified by comparing its results with the result of the conventional (single processor) simulator. The target simulation phase of our approach can be divided into the following three stages:

1. Testbench forwarding,
2. State restoring, and
3. The target, slice simulation stage.

The correctness of the first stage is assured by the stimulus from the dummy DUT. This stimulus is functionally the same as used by the conventional simulator, since it comes originally from the zero-delay model simulation. The timing of the stimulus is also the same as in the original simulation on the cycle boundary, provided that there is no timing violation in the design.

For the second stage, once all design state is restored correctly, combinational logic values are determined immediately. The third stage should also be correct because the target slice simulation uses the original DUT and the original testbench. Therefore, the correctness of temporal parallel simulation is easily ensured.

5. Performance Analysis

The following expressions represent the total parallel simulation time T_{tot} as a function of the number of simulation slices N and some overhead due to reference simulation, and saving/restoring of the testbench and DUT:

$$T_{tot} = T_{ref} + T_{target}$$

$$T_{ref} = C_{stim} \cdot T_{0-delay} + N \cdot T_{save}$$

$$T_{target} = T_{restore} + \frac{T_{timing}}{N} + T_{TB} \cdot \frac{X-1}{N}$$

where:

T_{tot} = total parallel simulation time,
T_{ref} = reference simulation time,
T_{target} = target simulation time for each slice,

C_{sim} = overhead due to stimulus capturing,
$T_{0\text{-}delay}$ = simulation time for conventional zero-delay model,
T_{save} = time to save the DUT state,
$T_{restore}$ = time to restore the DUT state,
T_{timing} = simulation time for conventional full timing model,
T_{TB} = simulation time for testbench forwarding,
X = index of the target simulation slice,
N = total number of simulation slices.

The total parallel simulation time is a sum of the reference simulation time and target simulation time. Note that only one slice is included in the target simulation time, since all slices are simulated in parallel. Reference simulation time is a function of conventional zero-delay simulation and some overhead. Overhead due to stimulus capturing is multiplied by the simulation time because capturing event occurs continuously during the entire simulation. Target simulation time is a function of some overhead and conventional full timing simulation time for particular slice, labeled X in the equation. Note that the testbench is simulated with dummy DUT from the simulation time 0 to the previous slice for the purpose of testbench forwarding.

Parameter T_{ref} is usually very small compared to the conventional timing simulation because it is a zero-delay simulation. It is known that zero-delay simulation is at least 10~50 times faster than full timing simulation. As the number of DUT output ports is very limited, stimulus capturing C_{sim} does not introduce a big overhead. Similarly, T_{save} is small. The value of T_{target} is also small because only a single slice is considered. Therefore, the total parallel simulation time T_{tot} is significantly smaller than the one of conventional simulation.

An important point is that the simulation time for each target slice is not identical. In other words, the last slice (X=N) has slowest performance because of the overhead for testbench forwarding. However, the performance difference is so small that it can be ignored because testbench is usually described on a high abstraction level; it executes hundreds of times faster than the full timing simulation of DUT. Hence the overhead of testbench is a relatively small fraction of the total simulation time. This small performance difference will be shown later in the experimental result section.

6. Experimental Results

The following experiments were performed using zero-delay gate-level simulation as a reference simulation.

6.1 Experiment 1

The experiment was carried out with the help of designers from a major semiconductor corporation on an industrial 18M-gate design. The design was elaborated using a commercial parser and simulated with Cadence NC-Verilog simulators, running on SUN machines. Despite a small (10:1) speedup ratio between the zero-delay simulation and full-timing GL simulation, the results showed an expected linear (6×) speedup with 10 simulators on a simulation run of 72,600,000 cycles.

6.2 Experiment 2

In this experiment an S1_Core design from Simply RISC [15] was used. The design was elaborated using a commercial parser and simulated with Cadence NC-Verilog simulators, running on PCs with Linux. The design has 1.2 million gates and contains one 64 bit-SPARC Core, a Wishborn bridge, a reset controller and a basic interrupt controller. The following shows the characteristics of the traditional, single-processor simulation and the zero-delay simulation used as reference simulation.

Single-processor Simulation (full GL timing)
- Simulation Size : 500,000 cycles
- Simulation Time : 11,860 s
- Simulation Rate : 42.16 cycles/sec

Zero-Delay simulation
- Simulation Cycle : 500,000 cycles
- Simulation Time : 223 s
- 53 × faster than full timing simulation

The results of the simulation are shown in Table 1. As discussed earlier (refer to the equations) the time to simulate individual slices during the target simulation differs slightly among the slices. This is due to the time needed to save/restore the testbench, with the restore time increasing with the number of slices. The best simulation performance is attributed to the first slice: the initial condition for the first slice does not require the design state to be restored, and the testbench doesn't need to be simulated to recover the state. From the second slice on, the simulation may be slower. However, the plots in Fig. 3 and Fig. 4 show that the overhead to simulate testbench is small and the last slice is simulated almost as fast as the first one. Furthermore, the simulation overhead due to testbench forwarding is not a function of the number of slices but a function of the simulation time. Therefore, regardless of the number of slices, the total simulation overhead of temporal parallel simulation is maintained at a small level.

Slice No.	First simul. (zero-delay)	Second simulation (full timing)			Total simulation time (min, max)
		1st slice (best case)	2st slice	Last slice (worst case)	
5	361	2673	2756	2798	3034 -159
10	391	1342	1554	1567	1733 - 958
15	429	887	951	969	1316 -1398
20	457	648	732	745	1150 - 1202

Table 1. Performance of gate-level simulation with 20 simulation slices. Design: S1Core, 1.2Mgates.

The table shows that up to 10× speedup improvement is achievable with 20 target slices, with the overhead for testbench and design state restoration taken into account. The performance improvement starts to saturate after 15 slices due to overhead incurred by reference simulation and state restoration. It should be noted that state restore overhead is around 7% of total simulation for slice No. 20. In general, the total simulation time is long enough to ignore this overhead allowing for large number of simulation slices in real verification.

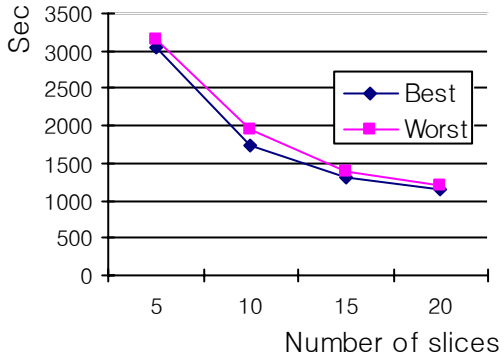

Fig. 3 Gate-level timing simulation time as a function of the number of simulation slices.

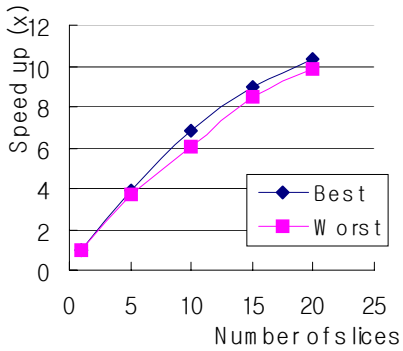

Fig. 4 Gate-level timing simulation performance as a function of the number of simulation slices.

7. Conclusions and Future Work

A radical solution to completely eliminate communication and synchronization overhead in a distributed parallel simulation environment for full timing gate level simulation is proposed. This is accomplished by performing temporal partitioning of the simulation run, instead of spatial partitioning of the design. For long simulation runs a linear speedup can be obtained; this is something that is not achievable in traditional (spatial) parallel simulation, due to the unavoidable inter-simulator communication and synchronization overhead. To achieve this linear speedup the implementation may have to be confined to a strictly progressive refinement process, in which lower abstraction model is cycle-time or transaction-time consistent with its

higher abstraction model. However, we believe that it is more than worthwhile because the temporal parallel simulation may make the fully automated and fast dynamic verification possible by integrating the verification flow with implementation flow.

The proposed approach is also applicable to designs with multiple asynchronous clocks. By applying proper delay on a clock domain crossing (CDC) wire during the reference simulation, we believe the consistency can be maintained for designs with multiple clocks. As a result, our approach might be applicable for any general designs, and this extension will be our further research items in the future.

REFERENCE

[1] Y. C. Hsu et al, "Visibility Enhancement for Silicon Debug," *Proc. 43rd Design Automation Conference*, pp. 13-18, July 2006

[2] Bauer, J. et al., ''A Reconfigurable Logic Machine for Fast Event-driven Simulation,'' Proc. ACM/IEEE DAC, pp. 668-671, June 1998.

[3] Fujimoto, R., ''Parallel Discrete Event Simulation,'' Communications of the ACM, Vol. 33, Issue 10, pp. 30-53, Oct. 1990.

[4] Ghenassia, F., *Transaction Level Modeling with SystemC*, Springer, Dordrecht, Netherlands, 2005.

[5] http://www.deepchip.com/items/0421-01.html

[6] Jayadev Misra, "Distributed Discrete-Event Simulation." ACM Computing Surveys, 18(1):39–65, 1986.

[7] A. Gafni. "Rollback Mechanisms for Optimistic Distributed Simulation Systems." Proceedings of the SCS Multiconference on Distributed Simulation, vol.3, pages 61-67, July 1988

[8] R.M. Fujimoto, "Time Warp on a Shared Memory Multiprocessor", Transactions of the Society for Computer Simulation, Vol, 6, No. 3, pages 211-239, July 1989

[9] R.M. Fujimoto, "Parallel Discrete Event Simulation," Communication of the ACM, Vol. 33, No. 10, Oct. 1990, pp 30-53.

[10] H. Bauer, C. Sporrer, and T. Krodel. "On Distributed Logic Simulation using Time Warp." In Proc. VLSI International Conference (IFIP), Edinburgh, 1991

[11] Tun Li; Yang Guo; Si-Kun Li "Design and implementation of a parallel Verilog simulator: PVSim" VLSI Design, 2004. Proc. 17th International Conference on 2004 pp.329 – 334.

[12] L. Zhu, G. Chen, B.K. Szymanski, C. Tropper, Tong Zhang "Parallel Logic Simulation of Million-Gate VLSI Circuits" Proc. 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems - Volume 00 MASCOTS '05, 2005. Also, MS thesis on the same subject (RPI, 2005).

[13] *SimCluster* datasheet, Avery Design Automation (http://www.avery-design.com)

[14] *MP-Sim* datasheet, Axiom Design Automation (http://www.axiom-da.com)

[15] S1_Core RISC processor, Simply RISC (www.srisc.com)