Temporal Parallel Simulation: A Fast Gate-level HDL Simulation Using Higher Level Models

Dusung Kim[1] Maciej Ciesielski[1] Kyuho Shim[2] Seiyang Yang[2]
[1]Department of Electrical and Computer Engineering
University of Massachusetts, Amherst, MA, USA 01003
{dukim, ciesiel}@ecs.umass.edu
[2]Department of Computer Engineering
Pusan National University, Busan, Korea, 609-735
{capnemo,syyang}@pusan.ac.kr

***Abstract*—Simulation speedup offered by distributed parallel event-driven simulation is known to be seriously limited by the synchronization and communication overhead. These limiting factors are particularly severe in gate-level timing simulation. This paper describes a radically different approach to gate-level simulation based on a concept of temporal rather than conventional spatial parallelism. The proposed method partitions the entire simulation run into simulation slices in temporal domain and each slice is simulated separately. With each slice being independent from each other, an almost linear speedup is achievable with a large number of simulation nodes. This concept naturally enables "correct by simulation" methodology that explicitly maintains the consistency between the reference and the target specifications. Experimental results clearly show a significant simulation speed-up.**

Keywords : Event-driven simulation; parallel simulation; verilog simulation; Gate-level simulation.

I. INTRODUCTION

Event-driven hardware simulation remains the most widely used technique for functional and timing verification, owing to its many advantages, and it will remain such for a foreseeable future. However, HDL simulation suffers from very low runtime performance, dictated by its inherently sequential nature. There have been several approaches to address this deficiency, such as modeling design at higher abstraction level, performing hardware-assisted simulation acceleration, or distributed parallel simulation. Even though some of the techniques have been successfully employed in industry, it is still hard to achieve a sufficiently high simulation speed to handle current complex large designs. In gate-level (GL) timing simulation, the problem is particularly severe. With the interest in gate-level timing simulation fueled by designs fabricated in nanometer device technology, there are many indications that industry will see gate-level timing simulation rampant [1].

To improve performance of gate-level timing simulation we introduce a new, radically different approach to parallel HDL simulation. The proposed method addresses some of the deficiencies of current distributed simulation. One of them is the design partitioning, which should minimize the inter-module communication and synchronization. Such a partitioning, which must work universally well for any design, is a known intractable problem, and a suboptimal partition strongly affects the performance of distributed simulation.

In contrast, the proposed method does not require design partitioning, so there is no communication and synchronization overhead imposed on simulation. The method consists of two major steps: (1) Fast reference simulation that runs on a higher (reference) level design model and collects the necessary information about the design state (i.e., register values and memory print); and (2) target simulation, running on a lower (target) design level, distributed to individual simulators. The entire simulation run is divided into slices, each to be executed on an independent simulator. For this reason, we refer to this technique as *temporal parallel simulation* (TPSim) in contrast to the spatial parallel simulation (a.k.a. conventional distributed parallel simulation). The basic idea of this approach and preliminary results for special cases were introduced in [2]. In this paper we describe a solution to some unresolved problems and generalize this approach to an arbitrary large design, resulting in a much better simulation performance.

After reviewing the state-of-the-art in this field, we outline the basic concept of our approach and discuss several practical issues. The experimental results demonstrate that our approach provides a dramatic performance improvement compared to the conventional simulation.

II. STATE OF THE ART

In order to increase performance of the simulation based verification some designers resorted to hardware-assisted simulation acceleration; in this approach the synthesizable portion of the design under verification (DUV) is emulated in hardware (HW), while stimulus is applied from the software HDL simulator [3]. In practice, however, performance of such HW-accelerated simulation is severely limited by overhead introduced by the testbench and the communication overhead between the testbench (residing in the simulator) and the design (emulated in hardware). As a result, HW-accelerated simulation can achieve at best a 10 fold speedup for complex designs when using signal-level testbench. In addition to high cost and a complicated hardware setup, this solution also suffers from long hardware compilation time, limited signal visibility and poor controllability of internal design points - something that is naturally supported by software HDL simulators.

978-3-9810801-7-9/DATE11/©2011 EDAA

Other approach to simulation is to use a more abstract design model, such as cycle-based simulation or transaction-level simulation based on transaction-level models (TLM) [4]. There are also attempts to translate the initial (RTL) design specification into C and simulate the design on that level using standard C compilers [5]. Designers use it in conjunction with formal verification, such as equivalence checking and model checking, which can verify certain design properties globally. However, because of large complexity of the underlying mathematical models, formal methods are still limited to relatively small portions of design or to specific design domains. Furthermore, neither hardware acceleration nor formal verification can efficiently solve the gate-level timing simulation.

Other methods rely on distributed parallel simulation, which partitions the design into separate modules and performs concurrent simulation using multiple HDL simulators [7]. A rich body of literature exists in the area of parallel simulation, known as *Parallel Discrete Event Simulation* (PDES) [6]. Chamberlain [8] discussed several issues related to this concept, such as partitioning, synchronization, and granularity. Fujimoto [6] and Nicol [9] intensively researched rollback-based and lock-step based synchronization in PDES.

Bagrodia et. al. [10] developed a parallel gate-level circuit simulator in the MAISIE simulation language and implemented it on both the distributed memory and shared memory parallel architectures, achieving speedup of 2-3× on eight processors. Lungeanu [11] proposed a "dynamic" approach, which combines conservative and optimistic approaches by switching between the two protocols depending on the amount of rollback. They demonstrated speedup of up to 11× on 16 processors on a circuit with 14k gates.

Li et. al. [12] claim to have developed the first Verilog distributed simulator even though they failed to get the desired performance improvement. Zhu et. al. [13] achieved a considerable speed up improvement with a large gate-level decoder design. However, such a design is a special case that provides almost ideal partitioning, which is generally not achievable.

Most of the results in this area have been demonstrated only on small to moderate-size, single-clock designs that can be partitioned without incurring significant inter-module communication and synchronization. Therefore, only a few commercial products have been developed, including SimCluster [7] and MP-Sim [14], the latter one requiring a proprietary simulator. However, they have not attracted the expected attention of designers, due to their limited performance and scalability. Most recently, the parallel gate-level simulation methods using GPU [15] have been proposed, but they are confined to gate-level with zero delay only. Furthermore, their performance strongly depends on the type of design being simulated.

III. TEMPORAL PARALLEL SIMULATION

A. Basic Idea

The temporal parallel simulation partitions the simulation run in time, by cutting the entire simulation period into a number of independent simulation slices. It consists of two major steps:

- Fast reference simulation, performed on a high-level abstraction of the design to store essential state information at selected checkpoints. This simulation is done on single processor.
- Detailed, fine-grain target simulation, performed on a lower level (gate-level) model. It is applied in parallel to each simulation slice, distributed to the individual simulators.

Fig. 1 illustrates the basic idea. For this approach to work, the initial design state for each slice of the target simulation must be first captured and saved during the first (reference) run.

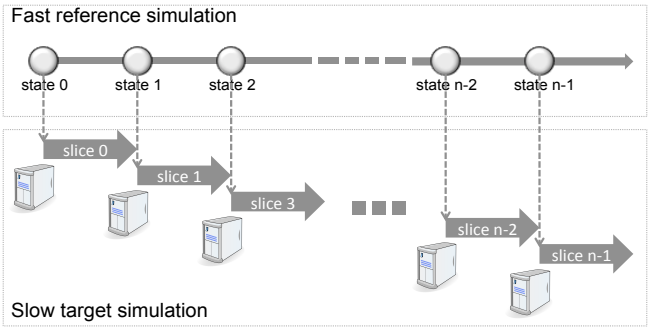


Figure 1. Concept of temporal parallel simulation

This is accomplished at predetermined checkpoints, determined by the number of processors available for 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.

The performance of this method, measured in total simulation time *T,* can be estimated as follows:

$$T = \sum_{i=1}^{n} T_{Ss}(i) + T_{Rsim} + \max[T_{Tsim}(i) + T_{Sr}(i)] : 1 \le i \le n \qquad (1)$$

where $T_{Ss}(i)$ is the state saving time for slice i; T_{Rsim} is the conventional simulation time for the reference model; and $T_{Tsim}(i)$ and $T_{Sr}(i)$ are the conventional simulation time and the state restoring time for one slice for the target model, respectively.

Since the overhead for state saving $T_{Ss}(i)$ and restoring $T_{Sr}(i)$ is considerably small, reducing T_{Rsim} is key to make this concept practical. For GL timing simulation, RTL model is a natural candidate for the reference simulation since the simulation of such model is more than 100 times faster than the corresponding GL timing simulation.

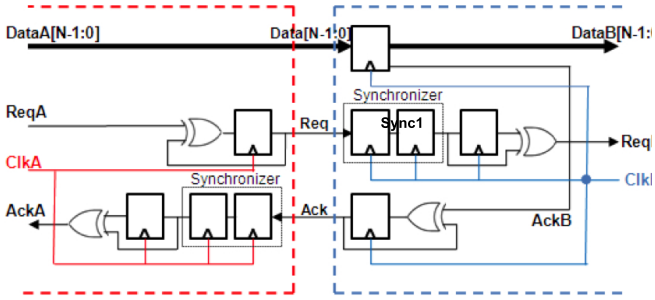
We should note that T_{Rsim} might not be counted towards the total simulation time (T) if such a simulation is mandatory and is carried out at the higher abstraction level during the common design implementation/verification flow. That is, the

simulation performed during the higher-level model verification, can serve as a reference simulation for temporal parallel simulation at a lower level, without additional overhead. Experimental results are shown in Section IV.

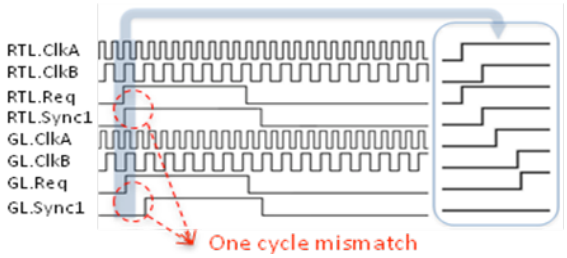
B. Difficulties in Generalization of Temporal Parallelism

1) Multiple Asynchronous Clocks

Contrary to a popular view, GL simulation for multiple-clock design may not be 100% cycle-by-cycle consistent with the RTL simulation, even if there is no timing violation. Fig. 2 illustrates this case with an example of typical two-phase handshaking logic.



(a) Two-Phase Handshaking Logic



(b) Timing mismatch in CDC

Figure 2. Two-Phase Handshaking Logic.

In Fig. 2(a), two synchronizers are used for Req and Ack signals, respectively. No synchronizer is used for Data because the signal values in data bus are maintained at the same value for sufficiently long time so that the receiving flip-flop can sample stable values. Fig. 2(b) shows timing inconsistency between RTL simulation and gate level timing simulation. In this case, flip-flop Sync1 in gate level simulation cannot sample value 1 on Req, which can be sampled in the corresponding cycle in RTL simulation. This is because the delay of Req makes the value change from 0 to 1 happen after the rising edge of ClkB. As a result, the sampling signal value of Req is delayed for one cycle. This inconsistency causes our approach to produce, in general, different simulation result from the conventional gate level timing simulation. Therefore, a simple state saving from RTL simulation and restoring into GL timing simulation does not work for designs having multiple asynchronous clocks

2) State Checkpointing in Event-driven Simulation

In cycle-based simulation, the checkpoints can be assigned at the end of any cycle period without causing any discrepancy between the reference and target simulation. In an event-driven simulation, however, finding correct placement for checkpoints is more difficult because of arbitrary delay between the event edges.

To illustrate this issue let us consider a fragment of Verilog code in Fig. 3, which is a part of the reference RTL model in TPSim. It has #1 (one unit) delay at the right hand side of the non-blocking assignments. This #1 delay models the clock-to-Q delay of the corresponding flipflops. In fact, there are many reasons that designers use such delays in their Verilog codes, e.g., for debugging convenience, mixed RTL/gate-level simulation, etc. [16]. The right side in Fig. 3 is a waveform from the actual simulation of the code. If the checkpointing is made at 300,001 nsec (*CP2*) of the simulation time, the correct value, 1, is saved and is restored later for target simulation. This is the correct behavior. However, if the checkpointing is made at 300,000 nsec (*CP1*), the incorrect value, 0, is saved instead. The wrong value is restored at the corresponding flipflop in the target simulation, providing a wrong starting point for the (target) simulation. Hence, the resulting simulation is incorrect. One possible solution is to ignore all delays that appear in the high-level abstraction of the design to be simulated for the reference simulation. But ignoring such delays in the high-level abstraction of the design may also result in an incorrect checkpointing, especially for designs with multiple asynchronous clocks.

```
always @(posedge clk)
begin
  if(!rst_n) q <= #1 0;
  else       q <= #1 d;
end
```

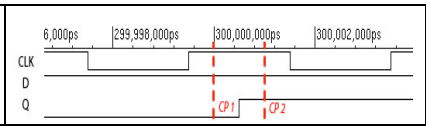


Figure 3. Example code showing problems arising in state checkpointing.

3) State Matching

Besides the timing issues mentioned above, one must maintain functional correctness of the restored target state. This in turn requires matching of the states in the RTL design with those in the GL design. While the states in RTL and GL models are represented by state registers, finding direct relationship between the registers is not always possible. This is because during synthesis the design undergoes a number of logic transformations, such as combinational and sequential logic optimization, retiming, and algebraic transformations. A promising preliminary work in state matching has recently been published in [17]. In current version of TPSim, such sequential transformations are not considered. We assume that there is either a direct one-to-one register relationship or other trivial relationships between RTL and GL registers (caused, for example, by bit truncation, removal of duplicated registers, etc.). Handling retimed design using technique introduced in [17] is planned as future work.

4) Handling testbench

While the design state of the DUT can be stored at any point during the reference simulation, the state of the testbench cannot be similarly captured, and the stimulus generated by the testbench cannot be restarted arbitrarily. This is because testbench is a sequential process that has no hardware "states", so it cannot be restarted at an arbitrary point of time.

C. *Proposed Solutions for Temporal Parallel Simulation*

In this section, we describe solutions and implementation issues to address the problems mentioned in section III.*B*.

As the design state is saved during the functional reference simulation (using e.g. RTL) and restored for the timing target simulation, timing discrepancies may appear at the beginning of each target slice in TPSim. An example of such a situation is given in Fig 4(a). The correct value of register *R1* at the checkpoint is 0 in timing simulation. However, corresponding saved value is 1, which is incorrect.

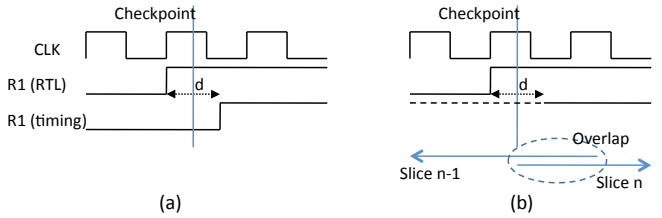
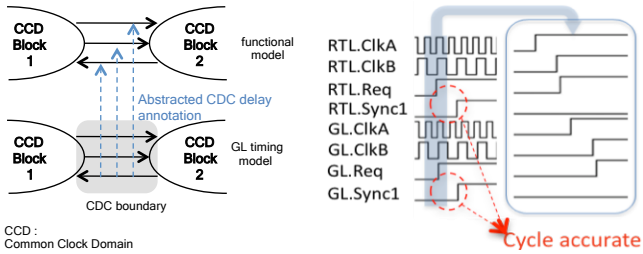


Figure 4. Initial state mismatch and slice overlapping

To address this issue, an overlap is created between two consecutive slices, as shown in Fig. 4(b). Consecutive slices, (*n-1* and *n*), are allowed to overlap by the value equal to the longest delay in the design. The correct timing simulation result for the overlap interval is generated from slice *n-1*. Every slice eventually must produce correct timing simulation result because the timing discrepancies cannot propagate across the clock cycle boundary.

For a design having multiple asynchronous clocks, we employ an *abstract delay annotation* method on top of the overlap approach. It is because such designs may not maintain the cycle-by-cycle consistency, as explained in section III.B.1. Fig. 5(a) explains this concept.



(a) Abstract delay annotation (b) Result for Abstract delay annotation

Figure 5. Abstract Delay Annotation

After analyzing the structure of Clock Domain Crossing (CDC), the delay information for the CDC boundary can be obtained from the Standard Delay Format (SDF) file. The information is recalculated and simplified in order to fit the RTL model. The new abstract delay is a function of the propagation delay for CDC boundary and clock skew between two asynchronous clocks, as given by the following equation.

$$D_{rel}(CDC) = D_{abs}(Clk_{send}) - D_{abs}(Clk_{recv}) + D_{abs}(CDC) \qquad (2)$$

Where D_{abs} is the absolute delay described in the SDF file; D_{rel} is the delay to be applied on CDC path only for the reference simulation; Clk_{send} is the clock for upstream Common Clock Domain (CCD); and Clk_{recv} is the clock for downstream CCD.

Fig. 5(b) shows that RTL.Req signal is also properly delayed after imposing proper abstract delay annotation. After annotating abstract delay to RTL design, the RTL reference simulation and gate level target simulation should be cycle-by-cycle consistent even in multiple-clock designs. Therefore, our approach produces identical result as conventional simulator unless there is a timing violation in the DUT. If there is timing violation, TPSim detects it efficiently by extending the overlap period beyond the overlap area, as explained in Fig 4. Such an extended overlap period is useful in detecting potential timing bugs. Since the simulation results for extended overlap period between consecutive slices should be identical if there is no timing violation, any inconsistency among those slices in that period clearly indicate potential timing bugs (e.g. set-up/hold time violation, glitches due to multiple combinational paths on CDC boundary). This is a very powerful feature, especially for the verification of multiple-clock design.

Generalizing this feature, the model at the higher abstraction level naturally plays a role of the reference model. Therefore, our method can automatically determine whether the design to be simulated is consistent with the model at the higher level of abstraction. Possible simulation mismatches between the model at the higher abstraction level and the model at the lower level can be automatically detected and reported to the designer or verification engineer for the possible investigation and debugging. Therefore, we believe that our temporal parallel simulation method naturally provides "correct by simulation" methodology that explicitly maintains the consistency among the models at the different levels of abstraction through the whole design process, once the first design model has met the specification.

To address checkpointing issues discussed in Section III.*B*.2, we define a *checkpoint window* as an interval dedicated to saving and restoring design state. The size of the checkpoint window is one clock-cycle equivalent. The three dotted boxes

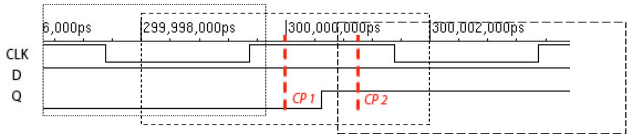


Figure 6. Checkpoint window

in Fig 6. represent the possible checkpoint windows for the case in Fig. 3. For every case, the correct value for Q could be reliably obtained at the end of each window because all signal transitions inside of each window (for a clock cycle) are reproduced during the state restoration in target simulation. Note that overlap period must be increased accordingly so that it contains the entire target checkpoint window.

The last issue that needs attention is handling the state of the testbench, as described in section III.*B*.4. Unlike in DUT, to recover the state at the restoring checkpoint, the testbench must

be simulated (executed) from the simulation time 0. In our implementation, we perform a fast testbench-only simulation to reach the target testbench state. We refer to this process as *testbench forwarding*.

Testbench forwarding is implemented as follows. The values of output ports of DUT, saved continuously during the reference simulation, serve as stimulus provider (a dummy DUT) for testbench simulation. The testbench is simulated with this stimulus from time 0 up to the starting point of the simulation slice in question. At this point the design state is restored form the data stored at the checkpoint, and the dummy DUT is replaced by the original DUT; each slice is then simulated normally and independently of the other slices. The experimental results in section IV show the the overhead for testbench forwarding is relatively small, compared to the total simulation overhead.

IV. Experimental Results

TPSim has been implemented with PLI as a plug-in for Cadence NC-Sim simulator. This made it possible to directly compare the simulation performance of the TPSim and conventional simulation. In the experiments TPSim was run with Cadence NC-Sim 8.2 simulator on an Intel T7500 CPU equipped computer. The target designs were synthesized by Design Compiler with TSMC 65nm technology library.

Experiment 1 – JPEG Encoder

In this experiment, we used JPEG Encoder design from OpenCores [18]. Total gate count of GL design is 0.9M. Table 1 shows the performance of TPSim for this design.

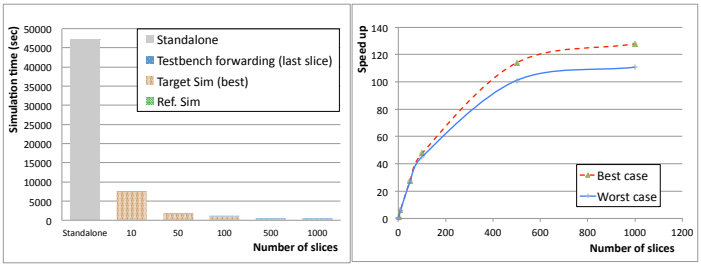
Table 1. Experimental results for JPEG Encoder

Design	Simulation Time (sec)	Ratio
RTL	184	1
GL timing	47192	X256

(a) Performance gap between RTL and GL timing simulation

# of slices	10		50		100		500		1000	
Ref. Sim (sec)	246		249		255		269		291	
	Best	Worst	Best	Worst	Best	Worst	Best	Worst	Best	Worst
Target Sim (sec)	7142	7191	1456	1508	737	791	145	198	78	135
Total (Ref. + Target) (sec)	7388	7437	1705	1757	992	1046	414	467	369	426
Speed up	6.38	6.35	27.7	26.9	47.6	45.12	114	101.05	127.9	110.78
TB f/w (Worst case) (sec)	56		State saving (sec)		< 0.5					

(b) Performance of TPSim



(a) Simulation time in TPSim (b) Speedup in TPSim

Figure 7. Performance of TPSim for JPEG encoder

Conventional GL timing simulation of JPEG Encoder is 256 times slower than RTL simulation. Under this condition, we were able to achieve speedup ranging from 6.35 to 110.78 times, depending on the number of simulation slices. Note that the worst-case target simulation refers to the simulation of the last slice, as it includes the longest testbench forwarding period. The speedup was based on the worst-case simulation as stated in equation (1).

As shown in Fig 7(a), TPSim has a linear speedup up to 100 slices and continues at a lower rate up to 500 slices. Beyond that point, the improvement tends to saturate but is still significant at 1000 slices. This is generally not possible with a conventional parallel simulation. We anticipate that a longer total simulation period with the same number of slices will delay the saturation point. This is because the target simulation period for the simulation run with 100 slices, shown in Fig 7(a), is too short, so that the reference simulation and testbench forwarding become dominating factors in the total overhead.

Experiment 2 – AES

In this experiment, we used AES design obtained from OpenCores. Total gate count of GL design is 25K.

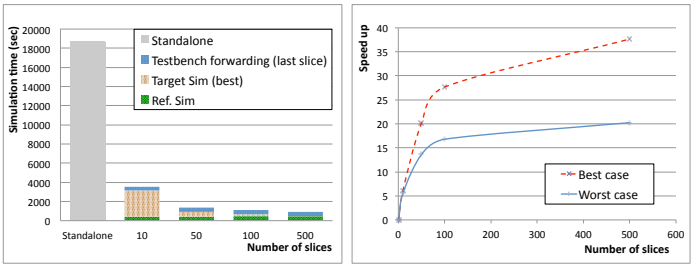
Table 2. Experimental results TPSim for AES

Design	Simulation Time (sec)	Ratio
RTL	110	1
GL timing	18669	X169

(a) Performance gap between RTL and GL timing simulation

# of slices	10		50		100		500	
Ref. Sim (sec)	426		431		442		453	
	Best	Worst	Best	Worst	Best	Worst	Best	Worst
Target Sim (sec)	2634	2916	498	940	233	670	43	469
Total (Ref. + Target) (sec)	3060	3342	929	1371	675	1112	496	922
Speed up	6.1	5.59	20.1	13.62	27.66	16.79	37.64	20.25
TB f/w (Worst case) (sec)	HDD				Exclude I/O			
	424				81			
State saving (sec)	< 1							

(b) Performance of TPSim



(a) Simulation time in TPSim (b) Speedup in TPSim

Figure 8. Performance of TPSim for AES

Table 2 shows that GL timing simulation of this design is 169 times slower than RTL simulation. In this case, the speedup ranges from 5.59 to 20.25 times. Fig. 8 shows a sub-linear speedup up to 70 slices. And the performance improvement continues up to 500 slices. However, considering

a large speed gap between the RTL and GL simulation, the overall speedup is lower than we would expect. In this case, however, the speed gap between RTL and GL simulation is small, because of the small size and low complexity of the DUV. Therefore, testbench forwarding overhead becomes relatively high. Fig. 8(a) shows that such factors dominate the entire overhead for the simulation with 50 slices. The optimum number of parallel nodes during this simulation period is 50. Table 2(b) also shows that reducing disk I/O overhead, by compressing data and using faster storage devices, will provide better results.

These two experiments demonstrate that TPSim offers higher performance improvement for designs having complex simulation data structure, longer simulation period and a large amount of event activities. Therefore, we anticipate that our approach will provide significant impact in dynamic verification of large-scale designs.

Experiment 3 – Cycle inconsistency in CDC path

Cycle inconsistency on some CDC paths between RTL and gate level requires abstract delay annotation for correct temporal parallel simulation (See Section III.*C*).

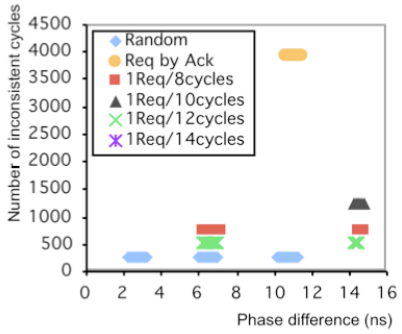


Figure 9. Cycle inconsistency on CDC path between RTL and GL simulation in two-phase handshaking logic in Fig. 2.

Fig. 9 illustrates the frequency of cycle inconsistency in two-phase handshaking logic in Fig. 2. The inconsistency on CDC paths between RTL and gate level simulation heavily depends on the clock phase relation and the frequency of *Req* signal. This is also true for other CDC logic. Therefore, providing an abstract delay annotation is important in TPSim for multiple-clock designs in order to handle the CDC problem so that RTL reference simulation and gate level target timing simulation become completely cycle consistent.

V. CONCLUSIONS

A radical solution to completely eliminate communication and synchronization overhead in a distributed parallel simulation environment for full timing gate level simulation is presented. This is accomplished by performing *temporal* partitioning of the simulation period, 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 an inherent overhead imposed by the inter-simulator communication and synchronization.

In addition, a helpful feature of our approach, is that it naturally provides reference comparison during the simulation, which is helpful in design debugging. Therefore, our approach provides not only significant performance improvement but also a smarter method for simulation-based verification.

ACKNOWLEDGMENT

This work was supported in part by the US National Science Foundation, award no. CCF 0702506 and CCF 1017530.

REFERENCES

[1] "Design Automation: Gate-level Timing Simulation Revs Up", Mar. 19, 2007. EETimes, < http://www.eetimes.com/showArticle.jhtml;jsessionid =U4GAX2JQT0AVUQSNDLOSKH0CJUNN2JVN?articleID=198000926>

[2] D. Kim, M. Ciesielski, K. Shim and S. Yang, " Temporal parallel gate-level timing simulation", *Proc. High Level Design Validation and Test Workshop (HLDVT)*, pp. 111–116, 2008

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

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

[5] Model Studio datasheet, Carbon. <http://carbondesignsystems.com>

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

[7] *SimCluster* datasheet, Avery Design Automation <http://www.avery-design.com>

[8] Roger D. Chamberlain, "Parallel Logic Simulation of VLSI Systems.", *Proc. 32nd ACM/IEEE Conference on Design Automation*, pp. 139-143, 1995.

[9] Nicol, David M. Principles of conservative parallel simulation. *Proc. of the 28th Winter Simulation Conference*, pp. 128-135, 1996.

[10] R. Bagrodia, Y. Chen, V. Jha, and N. Sonpar. "Parallel gate-level circuit sim- ulation on shared memory architectures." *Proc. of Computer Aided Design of High Performance Network Wireless Networked Systems*, pp. 170–174, 1995.

[11] Lungeanu, D., and Shi., C.J.R. "Parallel and distributed vhdl simulation." *Proc. Design, Automation and Test in Europe (DATE00)*, pp. 658-662, Mar. 2000.

[12] L. Li, H. Huang and C. Tropper, "DVS: An Object-Oriented Framework for Distributed Verilog Simulation", *Proc. of the 17th Workshop on Parallel and Distributed Simulation (PADS'03)*, 2003.

[13] 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 - MASCOTS* '05, 2005.

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

[15] Debapriya Chatterjee, Andrew DeOrio and Valeria Bertacco, "Event-Driven Gate-Level Simulation with GP-GPUs," *Proc. ACM/IEEE Design Automation Conference (DAC'09), San Francisco*, pp. 557-562, July 2009.

[16] Cummings, C. "Verilog nonblocking assignments with delays, myths and mysteries.", *Proc. Synopsys User Group Meeting (SNUG)*, 2002.

[17] D. Kim, D. Gomez, S. Yang and M. Ciesielski "Computing state matching in sequential circuits in application to temporal parallel simulation.". *Proc. International Workshop on Logic and Synthesis (IWLS).* pp. 171-177, June 2010.

[18] OpenCores, <www.opencores.org>