Parallel Multi-core Verilog HDL Simulation based on Domain Partitioning

Tariq B. Ahmad
ECE Department, UMASS Amherst
tbashir@ecs.umass.edu

Maciej Ciesielski
ECE Department, UMASS Amherst
ciesiel@ecs.umass.edu

ABSTRACT

While multi-core computing has become pervasive, scaling single core computations to multi-core computations remains a challenge. This paper aims to accelerate RTL and functional gate-level simulation in the current multi-core computing environment. This work addresses two types of partitioning schemes for multi-core simulation: functional, and domain-based. We discuss the limitations of functional partitioning, offered by new commercial multi-core simulators to speedup functional gate-level simulations. We also present a novel solution to increase RTL and functional gate-level simulation performance based on domain partitioning. This is the first known work that improves simulation performance by leveraging open source technology against commercial simulators.

Categories and Subject Descriptors

B.7.2 [**Integrated Circuits**]: Design Aids - *simulation, verification.*

General Terms: Verification, Simulation, Acceleration.

Keywords:
Simulation, Multi-core, ASIC, Opencores, RTL, Gate-level, Verilog.

1. INTRODUCTION

As the number, size and complexity of the designs continue to increase, so is the effort needed to verify the designs reliably and with good coverage. At the same time, reduced design cycle of 3-6 months makes verification an extremely challenging task. Today, verification consumes over 70% of the design cycle time and on an average, the ratio of verification to design engineers deployed is 3:1 [1][2].

Hardware description language (HDL) simulation is widely used for design verification, because of its ease of use, inexpensive computing platform, and 100% signal controllability and observability [3]. HDL simulation is used at all levels of abstraction, such as register transfer level (RTL), functional gate-level, and gate-level timing. As the design gets refined into lower levels of abstraction in the ASIC or FPGA design flow, such as gate and layout level, functional and timing simulations can validate the results of static timing analysis (STA) or equivalence checking (EC). Moreover, neither STA nor EC can find bugs due to X (unknown) signal propagation. Even though RTL regression is run on a daily basis, many design houses insists on gate-level simulation before sign-off [4].

The dominant technique used for functional and timing verification is HDL simulation [3]. However, HDL simulation suffers from extremely low performance because of its inherently sequential nature. As the design gets refined into lower levels of abstraction, and as more debugging features are added into the design, simulation time increases tremendously. Figure 1 shows the simulation performance of Opencores AES-128 design [4][5] at various levels of abstraction with debugging features enabled. As the level of abstraction goes down to gate or layout level and debugging features are enabled, simulation performance drops down significantly. This is due to large number of events at these lower levels, timing checks, and disk access to dump simulation data.

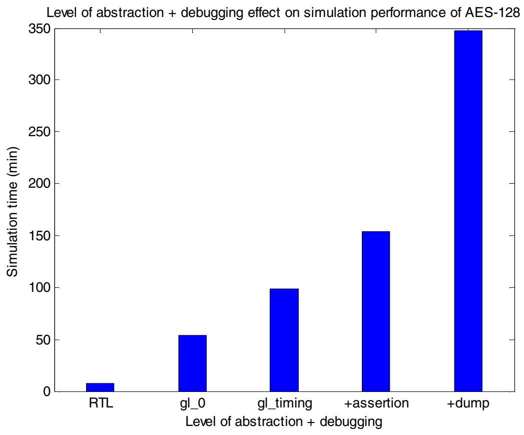
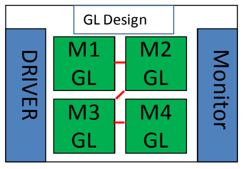


Figure 1. Drop down in simulation performance with level of abstraction + debugging

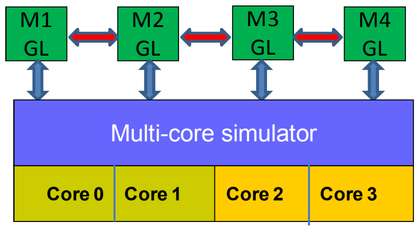
This work addresses the issue of improving simulation performance at RTL and functional (zero-delay) gate level. The next section discusses means for improving performance of functional gate-level simulation based on functional partitioning. In Section 3, we present a new approach of improving RTL and functional gate-level simulation performance based on domain partitioning. The paper is concluded in Section 4 and References are listed in Section 5. In the paper, we will refer to the *functional* gate-level simulation simply as gate-level simulation.

2. MULTI-CORE GATE-LEVEL SIMULATION BASED ON FUNCTIONAL PARTITIONING

A typical approach to parallel multi-core simulation, used by commercial simulator vendors, such as Synopsys, Cadence and Mentor Graphics, is based on functional partitioning. The term *functional partitioning* refers to the scheme where the design is partitioned into blocks (sub-functions) according to the function they perform in the design [6]. These sub-functions are then simulated on different CPU cores. Figure 2 shows a gate-level design which consists of four blocks M1 to M4. Each of these blocks represents a sub-function of the original gate-level design that can be considered as a partition. Commercial multi-core simulator can take these partitions and map them onto individual CPU cores as shown in Figure 2 (b).



(a) GL Model to be simulated



(b) Multi-core GL simulation

Figure 2. Functional partitioning based gate-level simulation

Figure 2 gives an illusion that multi-core simulation based on functional partitioning is scalable as one can further partition the design and map the design to as many CPU cores available on the computer. Unfortunately, this is not the case, and functional partitioning based multi-core simulation highly depends upon [7]:

a) Amount of inherent parallelism in the original design;

b) Partitioning scheme that preserves parallelism and load balancing;

c) Communication and synchronization overheads.

Communication overhead refers to data exchange between partitions during simulation. This is due to dependencies between the partitions. Synchronization overhead refers to the time needed to coordinate between the partitions. Partitions need to be synchronized with each other periodically before they engage in parallel activity [6]. It is worth mentioning that granularity in parallel computing refers to the ratio of parallel computation to communication [6].

Partitioning is a known NP-hard problem [7]. Given this, minimizing communication and synchronization overheads may pose conflicting requirements [8]. To illustrate the aforementioned points, we run functional partitioning-based multi-core simulations on a synthetically generated design consisting of a 128 bit combinational ripple-carry adder (RCA-128) at RTL and gate-level. The base design consists of a 128-bit RCA block and a testbench feeding stimulus to the adder. To create several partitions, the adder block is instantiated several times and chained as shown in Figure 3.

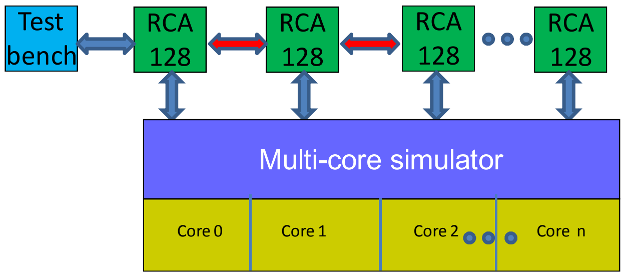


Figure 3. Experimental setup for measuring speedup in functional partitioning based multi-core simulation

Tables 1 and 2 show simulation speedup for RTL and gate-level simulations. All simulations are performed using Synopsys multi-core VCS Version H-2013.06 simulator on a quad-core Intel non-uniform memory access architecture (NUMA) machine with 8GB RAM.

Table 1. Simulation speedup at RTL

# of CPU Cores in use	# of design Partitions	Single core simulation time	Multi-core simulation time	Speedup
		t_{sc} (sec)	(Syn + com) overhead $t_{mc\_com+syn}$ (sec)	t_{sc} / $t_{mc\_com+syn}$
2	2	4	30	0.13
3	3	5	45	0.11
4	4	6	57	0.10
4	6	8	81	0.09
4	8	9	135	0.06

Table 2. Simulation speedup at gate-level

# of CPU Cores in use	# of design Partitions	Single core simulation time	Multi-core simulation time	Speedup
		t_{sc} (sec)	(Syn + com) overhead $t_{mc\_com+syn}$ (sec)	t_{sc} / $t_{mc\_com+syn}$
2	2	5	44	0.11
3	3	6	60	0.10
4	4	7	89	0.07
4	6	8	125	0.06
4	8	10	174	0.05

Tables 1 and 2 show speed degradation compared to a single core simulation, even though all the partitions have the same amount of activity (they are fully load-balanced). The design lacks inherent parallelism and granularity to yield significant speedup. This speed degradation becomes worse as the number of partitions are increased. This is due to excessive communication and synchronization overhead between the partitions.

In [7] authors consider the same RCA-128 circuit but add artificial parallelism in the adder circuit by repeating the add operation in a loop many times to increase the inherent parallelism in the adder. Even then, the design cannot scale due to communication and synchronization overheads despite ideal and load-balanced partitioning. The same authors also propose the idea of RTL prediction-based gate-level simulation [7],[8],[9] to reduce communication overhead between partitions as each partition locally generates the signals that it needs from other partitions. Even though this eliminates communication overhead, the synchronization overhead kicks in and this approach does not scale as well. Also, this prediction-based approach is only applicable to gate-level simulations and needs manual tweaking of the gate-level netlist. Unfortunately, the speedup gain from such manual effort is minimal. Not only that, the minimal speedup comes from designs that have inherent parallelism to begin with which renders this approach ineffective.

We ran another set of simulations on Opencores AES-128 gate-level design [5] which exhibits some inherent parallelism. Inherent parallelism in a design can be found by knowing the algorithm it implements or running simulation with a profiling option. Table 3 shows the AES-128 simulation profile. It shows low testbench activity compared to the design/module activity, which indicates its suitability for multi-core simulation based on functional partitioning.

Table 3. AES-128 simulation Profile

Most Active Module	**% Activity**
aes_sbox	24.2
aes_key_expand_128	12.4
testbench	3.9
aes_rcon	2.8
simulation overhead	9.8

When AES-128 gate-level is partitioned into 3 partitions based on grouping module instances, as shown in Figure 4, its simulation yields some speedup compared to single core simulation.

```
partition {test.u0.us03 (aes_sbox),
           test.u0.us10 (aes_sbox),
           test.u0.us22 (aes_sbox),
           test.u0.us23 (aes_sbox),
           test.u0.us30 (aes_sbox),
           test.u0.us31 (aes_sbox),
           test.u0.us32 (aes_sbox),
           test.u0.us33 (aes_sbox)
     };

partition {test.u0.u0 (aes_key_expand_128)};

partition {test.u0.us12 (aes_sbox),
           test.u0.us13 (aes_sbox),
           test.u0.us20 (aes_sbox),
           test.u0.us21 (aes_sbox),
           test.u0.us00 (aes_sbox),
           test.u0.us01 (aes_sbox),
           test.u0.us02 (aes_sbox),
           test.u0.us11 (aes_sbox)};
```

Figure 4. AES-128 partitioning based on module instances

Table 4 lists the simulation speedup as the number of partitions are increased. It shows that the best speedup occurs when the number of partitions equals three (partitioning shown in Figure 4). As the number of partitions increases beyond three, communication and synchronization overhead starts to dominate and speedup deteriorates.

Table 4. Multi-core simulation performance of AES-128 (Single core simulation time <u>*T1=160 min*</u>)

(AES-128) # of partitions	Partitioning Scheme	MC sim ***T2*** (min)	MC Speedup ***T1/T2***
2	instance-based	165	0.96
3	instance-based	125	1.28
4	instance-based	142	1.12
6	instance-based	144	1.11
8	instance-based	139	1.15

We conclude that functional partitioning-based gate-level simulation, as implemented in new commercial multi-core simulators, <u>*does not scale*</u> with the number of cores. It is extremely difficult to satisfy all the conditions such as load-balancing, inherent parallelism in partitions, low communication and synchronization overheads. For these reasons, functional partitioning-based multi-core simulation has not taken off, except for very specific designs. Also, RTL simulation based on functional partitioning is not successful because of fine granularity of RTL partitions. For this reason, commercial multi-core simulator vendors like Synopsys [12] and Cadence [13] limits multi-core simulation to only gate-level designs. This also serves as motivation to explore new avenues to run multi-core simulation such as based on domain partitioning.

3. MULTI-CORE RTL AND GATE-LEVEL SIMULATION BASED ON DOMAIN PARTITIONING

3.1 Introduction

As mentioned earlier, commercial simulators allow only functional partitioning. In this type of partitioning, the focus is on the computation that needs to be performed, rather than on the data provided for the computation. For designs that do not preserve design hierarchy or are flattened for optimization, functional partitioning fails. In contrast, a partitioning scheme is possible that relies on partitioning of the data instead of the function. Such a partitioning is called *domain partitioning* [6]. Figure 5 illustrates this concept.

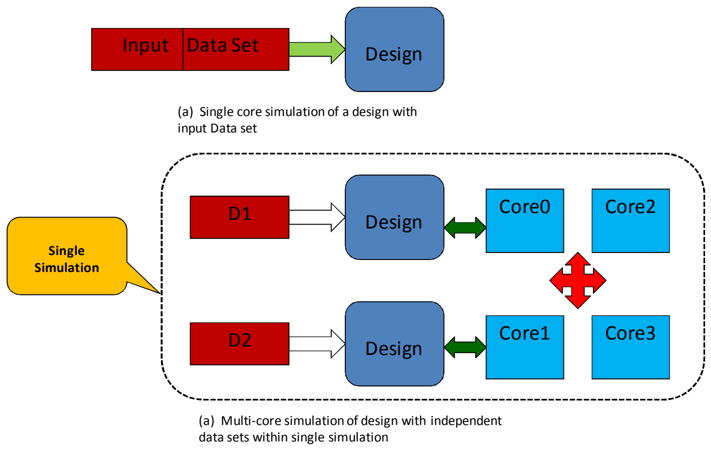

Input
Data Set
Design
(a) Single core simulation of a design with input Data set
Single Simulation
D1
Design
Core0
Core2
D2
Design
Core1
Core3
(a) Multi-core simulation of design with independent data sets within single simulation

Figure 5. Multi-core simulation based on domain partitioning

Figure shows input data set is partitioned into data sets D1 and D2. D1 and D2 are fed to the design in parallel within a ***single simulation***. Inside ***single*** simulation, one thread simulates the design with D1 on one processor core and another thread simulates the design with D2 on a second processor core. In this scheme, single simulation becomes multithreaded simulation with independent data assigned to each thread. *This is achieved without manual partitioning or without running two simulations.* Commercial simulators do not allow exploiting this kind of parallelism within a single simulation. Mimicking the proposed multithread simulation idea on a commercial simulator means running more than one simulation (and hence using more simulator licenses). This is an expensive proposition for companies and researchers with limited budget. Moreover, there is no automated way of doing domain partitioning-based multi-core simulation on commercial simulators.

In order to instrument domain partitioning for simulation, one needs to access simulator's internal data structures. However, commercial simulators do not allow this kind of access. For this reason, we resorted to an open source Verilog simulator, known as Verilator [10]. Verilator translates Verilog HDL into C/C++ code and then compiles the C/C++ code to generate simulation executable. Verilator has gained a lot of popularity and is being used across the EDA industry by major companies [14].

Once Verilog code is converted into C/C++, there are various libraries available for parallelizing C/C++ source code. We resorted to Open Multiprocessing (OpenMP) library [11]. OpenMP is an application programming interface (API) library for parallel programming shared-memory machines using C/C++ or Fortran. It is relatively easily to perform parallel programming using OpenMP as its syntax is easy and requires only a few changes to convert a serial program into a parallel program. The other major libraries are Posix threads (Pthreads) which require manual effort for parallel programming; and Message Passing Interface (MPI), which is primarily used for programming distributed memory systems. Figure 6 shows the flow of multi-core simulation with domain partitioning using Verilator + OpenMP.

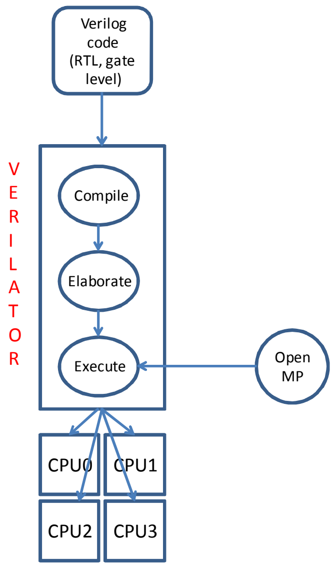
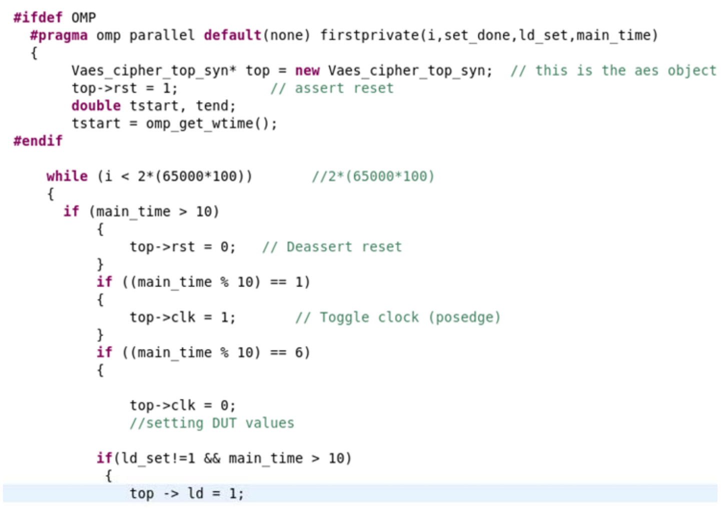


Figure 6. Proposed flow for multi-core simulation based on domain partitioning using Verilator and OpenMP

The C/C++ code produced by Verilator simulates design under test (DUT) only. Testbench is converted to C/C++ or SystemC using some of the constructs provided by Verilator. This step is semi-manual as majority of the constructs are provided by Verilator. Once the testbench is complete, OpenMP pragmas are inserted wherever testbench data can be partitioned. We used ***pragma omp parallel*** for AES-128 design's testbench. A snippet of the testbench with omp pragma is shown in Figure 7. *Firstprivate* refers to the variables that are private for each thread.



```
#ifdef OMP
  #pragma omp parallel default(none) firstprivate(i,set_done,ld_set,main_time)
  {
        Vaes_cipher_top_syn* top = new Vaes_cipher_top_syn;  // this is the aes object
        top->rst = 1;           // assert reset
        double tstart, tend;
        tstart = omp_get_wtime();
#endif

     while (i < 2*(65000*100))       //2*(65000*100)
     {
       if (main_time > 10)
           {
               top->rst = 0;   // Deassert reset
           }
           if ((main_time % 10) == 1)
           {
               top->clk = 1;       // Toggle clock (posedge)
           }
           if ((main_time % 10) == 6)
           {

               top->clk = 0;
               //setting DUT values

           if(ld_set!=1 && main_time > 10)
            {
               top -> ld = 1;
```

Figure 7. Inserting OpenMP pragma in the testbench to parallelize testbench.

3.2 Experiments

The performance of Verilator is further improved by adding parallelization using OpenMP. The combination of the two offers the best parallel HDL simulator, capable of handling RTL and gate-level designs. Tables 5-6 show simulation performance of AES-128 and RCA-128 at RTL and gate-level, respectively. All simulations are performed on Intel quad core non-uniform memory access (NUMA) machine with 8GB RAM.

Table 5. RTL and Gate-level simulation of AES-128 with 6500,000 vectors using Verilator and OpenMP

Number of threads	RTL Wall clock Time (sec)	Gate-level Wall clock time (min)	RTL Speedup /367	Gate-level Speedup /126
1	367	126	1	1
2	186	63	1.97	2
3	126	43	2.91	2.93
4	128	46	2.86	2.73
6	117	41	3.13	3.07
8	111	38	3.30	3.31

Table 6. RTL and Gate-level simulation of RCA-128 with 6500,000 vectors using Verilator and OpenMP

Number of threads	RTL Wall clock time (sec)	Gate-level Wall clock time (*min*)	RTL Speedup /19	Gate-level Speedup /11
1	19	11	1	1
2	23	6	0.82	1.83
3	16	3	1.18	3.66
4	14	4	1.35	2.75
6	12	3	1.58	3.66
8	9	2.8	2.11	3.92

Tables 5-6 show parallel simulation using Verilator and OpenMP to speed up simulation. The speedup is not linear, nevertheless it is significant. As the number of threads increases, synchronization overhead starts to dominate and makes the speedup saturate. It must be observed that speedup reported in Table 5 is much better than reported in Table 4. Hence, automated domain-partitioned based multithreaded multi-core simulation is worth on multi-core compute platforms.

3.3 Dependencies in the Testbench

There are designs, where testbench cannot be partitioned as shown in Section 3.1. Such a testbench is reactive, where the state of the testbench depends upon the state of DUT. We experimented with such a design to see how its performance degrades when simulated in parallel. We took AES-128 design and configured it such that one of its output feeds back into one of the inputs. This causes dependency as one cannot encrypt two plaintexts in parallel because the second plaintext needs the output of the first one. It was observed that despite dependencies, the performance of the design was not worse than a single threaded simulation. Hence, in the presence of dependencies, OpenMP still keeps the performance comparable to single threaded simulation. Note that this is not the case with functional partitioning where dependencies cause performance degradation, which is worse than running single core simulation.

3.4 Comparison with Commercial Simulator

Figure 8 compares the multi-core performance of Verilator and Synopsys VCS [12] simulator for AES-128 design. The design is functionally partitioned into four partitions for multi-core VCS simulation. For Verilator, the design is domain partitioned into four threads. Figure clearly shows Verilator performs better than VCS in multi-core simulation. Hence, domain partitioning is worth exploring and parallelizing for design that exhibit this kind of parallelism.

4. CONCLUSION

This paper explores two types of partitioning schemes for multi-core simulation of Verilog HDL designs. One scheme, based on functional partitioning, is currently being offered by commercial multi-core simulator vendors [12],[13]. We explored this scheme in greater detail and concluded that simulation performance does not scale with this scheme and new issues surface when other issues are taken care of. There is no way to reduce synchronization overhead in this scheme which increases with the number of partitions. Also, this scheme does not work for RTL simulation.

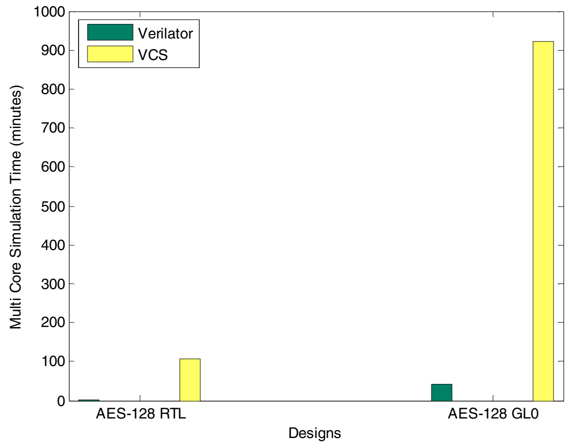


Figure 8. Multi core simulation performance comparison of Verilator and VCS at RTL and gate-level.

Currently, industry is in dire need of solutions to accelerate RTL simulation as such a simulation is done much frequently compared to gate-level simulation. Recently proposed scheme of RTL prediction-based gate-level simulation [8],[9] is based on functional partitioning, manual driven and speeds up only those designs that have inherent parallelism to a minimal extent, hence it is not effective.

The other scheme, based on domain partitioning [6], is explored in this paper. As commercial simulators do not allow access to simulator's source code, we used Verilator, an open source Verilog simulator, to instrument domain partitioning. We have not only successfully run single core simulations with Verilator but also added parallel processing to support domain partitioning using OpenMP. This is the first known complete solution to accelerate RTL and gate-level simulations targeting multi-core machine. As a next step, we are working on more and even larger designs to see the benefit of the proposed approach. This work has thus far received enormous interest by the industry and academia.

ACKNOWLEDGMENT

This work was supported in part by funding from the National Science Foundation, award No. CCF 1017530.

5. REFERENCES

[1] T. Anderson, and R. Bhagat, "Tackling Functional Verification for Virtual Components," ISD Magazine, pp. 26, November 2000.

[2] P. Rashinkar, and L. Singh, "New SoC Verification Techniques," Abstract for tutorial, IP/SOC 2001 Conference, March 19, 2001.

[3] W.K. Lam, "Hardware Design Verification: Simulation and Formal Method-Based Approaches," Prentice Hall, 2005.

[4] Tariq B. Ahmad, and M. Ciesielski, "Fast STA Prediction-based Gate-level Timing Simulation," Design and Test Europe, (DATE 2014).

[5] Rudolph Usselmann, "AES-128 Opencores design," Available at: http://opencores.org/project,aes_core

[6] Blaise Barney, "Introduction to parallel computing." Lawrence Livermore National Laboratory 6.13 (2010): 10.

[7] Tariq B. Ahmad, and M. Ciesielski, "An Approach to Multi-core Functional Gate-level Simulation Minimizing Synchronization and Communication Overheads," Microprocessor Test and Verification Conference, (MTVCON 2013).

[8] T. B. Ahmad, N. Kim, B. Min, A. Kalia, M. Ciesielski, and S.Yang, "Scalable Parallel Event-driven HDL Simulation for Multi-Cores," Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design, (SMACD 2012).

[9] D. Kim, M. Ciesielski, and S. Yang, "A new distributed event-driven gate-level HDL simulation by accurate prediction," Design and Test Europe (DATE 2011), pp. 547-550, March 2011.

[10] Wilson Synder, Duane Galbi, and Paul Wasson. "Introduction to Verilator," (2009).

[11] Dagum, Leonardo, and Ramesh Menon. "OpenMP: an industry standard API for shared-memory programming." Computational Science & Engineering, IEEE 5.1 (1998): 46-55.

[12] Synopsys VCS Verilog Simulator. Available at: www.synopsys.com/vcs

[13] Cadence Incisive Verilog Simulator. Available at: www.cadence.com/products/fv/enterprise_simultor

[14] Wilson Snyder, "Verilator: Fast, Free, But for me?" DVClub Presentation 2010, pp 11. Available at: http://www.veripool.org/papers/Verilator_Fast_Free_Me_DVClub10_pres.pdf