# Fast and Scalable Gate-level Simulation in Massively Parallel Systems

Haichuan Hu[1,2] Zichen Xu*[2] Yuhao Wang[2] Fangming Liu[1,3]
[1]National Engineering Research Center for Big Data Technology and System,
Services Computing Technology and System Lab, Cluster and Grid Computing Lab,
School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, 430074, China
[2]School of Mathematics and Computer Science, Nanchang University, Nanchang, China
[3]Pengcheng Laboratory, Shenzhen, China

***Abstract*—The natural bijection between a proposed circuit design and its graph representation shall allow any graph optimization algorithm deploying into many-core systems efficiently. However, this process suffers from the exponentially growing overhead and heavy memory footprint with the signal propagation. To conquer the unique challenge, we systematically study the simulation with millions of gates, and identify that the processing complexity could grow exponentially from the signal inputs, the skewness of the computational graph stays. Thus, we present ZhouBi, a fast and scalable gate-level simulation framework to fully exploit the parallelism from many-core systems. ZhouBi contributes in threefolds, (I) a graph representation that colors gate-level netlists and identifies skew partitions based on the graph skewness; (II) A set of heuristic algorithms that picks opportunistic and conservative algorithms to accelerate the simulation; (III) A system facility that supports selective mapping between simulation and many-core, providing a tradeoff between the risk of concurrent simulation fail and performance gain. We have prototyped ZhouBi and evaluated it with practical baselines. ZhouBi can achieve a 27.6× performance gain, as compared to the state-of-the-practice Veriwell without compromising any correctness. Our framework supports large graphs enabling scale-out gate-level simulations for chip design.**

## I. Introduction

Gate-level simulation plays an important role in designing modern chips. A smooth and correct gate-level simulation shall have a well-defined formal representation such that it could improve the cost-effectiveness for the circuit simulation. When the underlying circuit design scales, the complexity of the simulation grows exponentially, requiring some significant optimization efforts on both simulation software and supporting computing architecture. Some modern simulation tools include, Iverilog [1], Veriwell [2], Cadence [3], etc.

These computationally expensive software suffer from insufficient architecture support. It is desire to make a full use of the existing hardware parallelism and high-efficiency computing capability of the hardware [4]. However, the gate-level simulation is a flow of operations with temporal and spatial dependency. Naïvely distributing the simulation threads onto a many-core system challenges the simulation correctness.

In old days, to better understand the problem, we use graph to present the signal flow between gates [5]. In this graph representation, nodes represent the computing units, i.e., logic gates, and directed edges represent the logic dependencies in the temporal/spatial order. The gate-level netlists inherently have a bijective relationship with their graph representations. The graph allows the gate-level netlist grow elastically by adding/removing nodes. However, with a continuous input of circuit signals, a static graph could be duplicated and expanded in a temporal manner, leading a huge simulation graph with millions of nodes and edges. The graph cardinality grows in orders of magnitudes, compared to the original netlist graph, making a challenge task to efficiently simulate a very large gate-level graph. It is a challenge to improve the simulation efficiency while preserve the simulation correctness in a graph representation at scale out.

Accelerating the graph simulation is not a simple job *per se*. In past decades, many researchers have investigated in adopting parallelism from the underlying hardware. On one hand, for a complete simulation, some *conservative* techniques, such as the Chandy-Misra-Bryant (CMB) algorithm and many variants [6]–[8], are proposed to parallelize the simulation within a fixed multiplexing degree, according to the number of CPU sockets and cores. These optimization algorithms allow the multiple logical processes step forward at the same pace and block at many specific checkpoints to conservatively check the correctness within all time granularity, thus providing the correctness guarantee but lacking of the performance improvement potential. These techniques fail to harness the benefit of modern many-core architecture.

Some recent approaches opportunistically explore similar computational paths together in the graph [9]–[11]. These speculative algorithms allow multiple logical processes to proceed without considering the dependency between these processes. If the processing hits the jackpot, one may complete the whole simulation for many paths in the graph until it hits an error. The error forces the whole process rollback to the last correct checkpoint. These opportunistic algorithms advances by skipping the null message passing overhead from the conservative algorithm, thus are scalable. Yet these opportunistic algorithms suffer from the risk of a large overhead from rollbacks. For example, when the netlist graph is dense, i.e., a strong connective graph, the risk of potential performance degradation from rollback is unbounded. Thus, the optimization asks for a scheduling, *whether we can provide a leverage, between the conservative and opportunistic algorithms, to process a very large gate level simulation as a graph?*

To understand this problem, we have performed a system study on many simulation benchmarks, with a focus on the

performance on many-core systems. Our study reveals that (1) different circuits have different sensitivities to the two types of algorithms; (2) the ratio between a graph and its core part, i.e., graph skewness, stays with the input growth and is the invariable feature. The graph skewness is the sum of all minimum numbers of edges of connected components whose removal results in a planar graph [12], which further indicates the degree of dependence in a sub graph and its distribution. Based on the skewness, we further use the graph perfectness to theoretically cap the maximum cardinality of all sub graphs, thus better estimate the possible computational cost. With these indicators, we could only perform conservative algorithms on strongly dependent partitions, i.e., cliques, while opportunistically scale out on the weakly dependent partitions with estimated budgets, such that we could exploit the skewness of the circuit graph and adopt a proper computation part onto the underlying core(s). As such, we build a fast and scalable simulation framework by mapping graph partitions onto different algorithms and controlling the computation resource management.

In this paper, we present ZhouBi, a fast and scalable graph-represented simulation framework. ZhouBi achieves fast gate-level simulation at scale based on three facilities: (1) ZhouBi converts the synthesized gate-level netlist into a graph, and partitions the graph based on the graph skewness; (2) ZhouBi maps two algorithms, i.e., the conservative algorithm and the opportunistic algorithm, onto partitions with different skewness. (3) ZhouBi builds a system facility to assign each partition run to a running core, and allocates sufficient resources for conservative and opportunistic algorithms, respectively, balancing device utilization and maximizing efficiency. In this way, ZhouBi gains the benefit from both the conservative and opportunistic algorithms, while gaining advantages from modern architecture. We have prototyped ZhouBi as a gate level circuit simulator and evaluated it with many real-world benchmarks. Across different cases, ZhouBi exhibits a 27.6× and 25.9× performance improvement, as compared to the state-of-the-practice Veriwell [2] and iVerilog [1], respectively. Our main contributions can be summarized as follows:

- We study the performance problem of gate-level simulation using graph, and propose a framework ZhouBi to solve it.
- We identify the graph skewness and perfectness are key metrics to partition the netlist graph and estimate related cost, respectively.
- We schedule conservative and opportunistic algorithms based on graph skewness, harvesting merits from both algorithms and providing a tradeoff in-between.
- We carefully implement ZhouBi to deploy the two parallel versions of algorithms onto many-core systems, providing a holistic solution.
- We have evaluated ZhouBi with standard benchmarks, and show it can achieve a 27.6× performance gain.

The remainder of this paper is organized as follows. Section II highlights the background and our motivation on the gate-level simulation. Section III describes ZhouBi system design. Section IV empirically studies ZhouBi performance. Section V provides a brief on the related work. At last, Section VI concludes the paper.

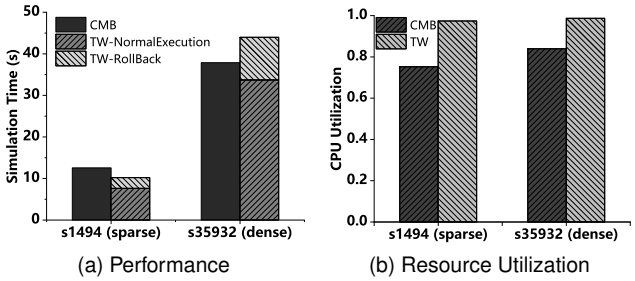

Fig. 1. Performance and resource utilization of the two simulation algorithms.

## II. Background and Motivation

In the aforementioned problem statement, we highlight that it is necessary to better understand the simulation graph of the gate-level netlist, or netlist graph, to help us better select the right simulation algorithm upon, while mapping it to the correct computation resources. In this section, we conduct such studies and collect our observations from a graph perspective.

### A. Affinity of Parallel Simulation Algorithms

The circuit parallelization is not a simple task, as the circuit has a complex correlation between gates. Previous work shows that there are generally two ways to ensure the consistency of the results in circuit parallel simulation: 1) the conservative algorithm, represented by CMB algorithm [6]; 2) the opportunistic algorithm, represented by TimeWarp (TW) algorithm [11]. The graph sparsity leads to different sensitivities to the two algorithms. We select a sparse graph and a dense graph on the classic IWLS benchmarks [13] to explore the affinity of the two original algorithms, as shown in Figure 1. We assign two CPU cores to simulate one thousand iterations of *s1494* and two hundred iterations of *s35932*. We find that, for sparse graph *s1494*, the opportunistic algorithm performs better; for dense graph *s35932*, the conservative algorithm performs better. These two types of algorithms are like two poles on the number line. The conservative algorithm guarantees correctness of the simulation, but at the expense of possible speedup, that is, the CPU utilization is low. The premise of the opportunistic algorithm is the shortest path based on graph theory. Each logical unit is simulated independently first, and the rolling back mechanism is required for the occurrence of errors. The denser the graph is, the greater risk of failures thus rolling back. Therefore, it prompts us to design a trade-off mechanism that allows us to take full advantage of both opportunistic and conservative algorithms using graph theory.

### B. Signal Propagation in Simulation Graph

There is a natural bijection relationship between the gate-level circuit and the graph representation, many scholars naturally apply any graph simulation algorithm to the circuit simulation after graph representation [14]–[16]. However, the

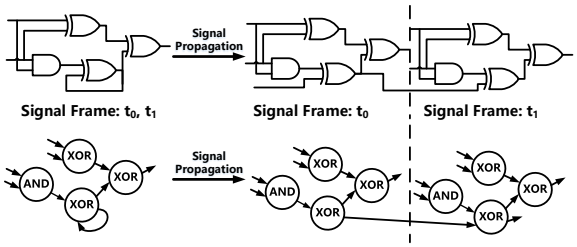


Fig. 2. An example of the signal propagation.

circuit simulation performance of graph representation is generally limited by the following two aspects: 1) the exponential growth of the simulation excitation signal [17]; 2) the graph structure represented by the graph is different from other common information storage, which is particularly resource-consuming [18]. The simple input signal becomes a shackle to graph-based gate level simulation. For example, as shown in Figure 2, the original gate level circuit graph has only 4 nodes and 9 pins. However, the input signals make the original graph generate many simulation copies. With the input signal pulse over time, the number of pins in the circuit graph increases linearly with the number of frames of the input signal. When the number of frames of input signal is $n$, the scale of the original graph could be expanded by $n$ times, and the complete simulation graph with $4n$ nodes and $9n$ pins could be obtained. Studies have shown that for a combinational gate level circuit with $N_{in}$ primary inputs, $N_{out}$ primary outputs, and $N_g$ logic gates, the number of test patterns required is $2^{N_{in}+N_g}$ in exhaustive simulation [19]. The size of the circuit expands linearly with the input of the signal. The total complexity is then $O(N_t \times 2^{N_{in}+N_g})$, where $N_t$ is the number of excitation signal frames. The time complexity of simulation increases exponentially with the growth of the graph size.

### C. Invariable Features of Circuit Graph

In terms of graph structure, the scale of the input signal graph changes, but some structural features of the graph remain unchanged: graph skewness and graph perfectness. The key reason behind why we cannot always opportunistically perform a TimeWarp algorithm for simulation is that some connected components in the graph are dense such that signal dependence in-between could be complicated. In this case, any misses in such dense subgraph could lead to catastrophic failure. These full-connected components are complete graphs, or *cliques*. To identify the number of the maximum clique in a graph, we adopt the concept of graph skewness, as the sum of every minimum number of connected component edges to a planar graph [12]. As circuit graph grows with the signal input, such skewness stays no change as the definition. This provides a unique feature of identifying the basic simulation unit for parallel processing. The clique serves as one encapsulated node in shrinking the scale of the graph, as shown in Figure 3.

To better understand the graph representation of gate-level netlist, we perform a system study on several standard benchmarks, such as IWLS benchmarks [13], EPFL benchmarks [20], and a decoder whose structure is similar to the

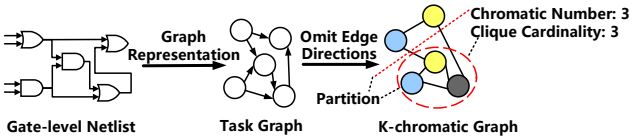


Fig. 3. Cliques and their cardinalities in the system study.

Viterbi Decode [21]. The graph statistics of these benchmarks are shown in Table I, featuring the graph skewness and perfectness. Though the cardinality of the graph varies, the two ratios can be calculated without scaling with the graph cardinality. The skewness feature is our knob to select a more suitable algorithm for specific graph partitioning. The graph perfectness tells the ratio between the largest chromatic number and the clique number. This is the bound of all dense sub graphs that are not suitable for an opportunistic simulation, thus the bottleneck of the computation (more details are discussed in Section III). In the next section, we further illustrate the design and implementation of ZhouBi.

## III. Design and Implementation

ZhouBi is a fast and scalable graph-represented simulation framework. The acceleration obtained with ZhouBi is achieved in a two stage approach. At the offline stage, ZhouBi inputs the gate-level netlist, converts it into a graph, omits the direction info and obtains a undirected graph. Further, ZhouBi computes the maximum chromatic number in all connected components in this undirected graph, and collects maximum cliques, i.e., every two distinct vertices in this subgraph are adjacent, at all chromatic levels. By computing the max cliques, the process identifies the skewness of the graph, and partitions the graph accordingly, sorted by the chromatic number. At the online stage, a scheduling module in ZhouBi revokes the graph representation with direction info, picks the conservative CMB algorithm for simulations in clique, then performs the TimeWarp algorithm to combine simulation between cliques. In this way, ZhouBi shuffles the computation from each clique and maps it onto different active cores. All results are simulated and broadcast in a cone-like partition broadcast since the signal dependence between cliques are weak. The ZhouBi architecture is shown in Figure 4. ZhouBi aims at gate-level simulation with the following assumptions:

- ZhouBi focuses on simulating gate-level netlist at scale using a generic graph representation with skewness.
- ZhouBi aims at higher level models, thus assuming there is no parasitic effect such as noise and crosstalk in the simulation.
- Without loss of generality, ZhouBi assumes the delay between input signal is a constant.

In this section, we introduce our design and implementation of ZhouBi. We discuss how ZhouBi recursively schedules different simulation algorithms to the right subgraph structure. Note that, as aforementioned in our study, most circuits exhibit a feature of small cliques. Thus, we partition the graph into small cliques, perform simulation, and merge/reduce them recursively. The CMB algorithm provides a bounded performance with correctness guarantee. We further provide an

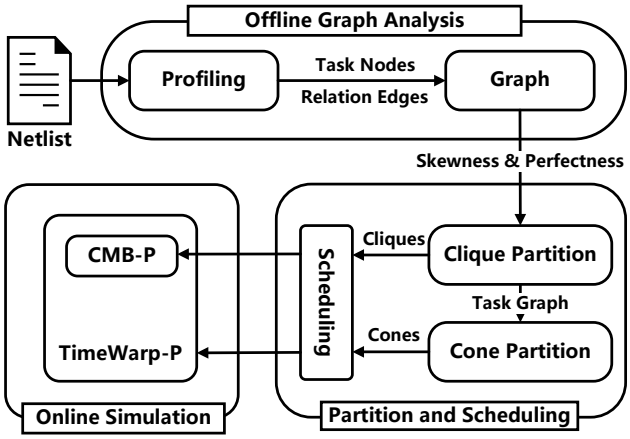


Fig. 4. The system overview.

extended version of CMB, called CMB-P, in ZhouBi to exploit the many-core feature of the underlying architecture. Similarly, for the simulation in cone, ZhouBi also provides a parallel version of the TimeWarp algorithm, known as TimeWarp-P, to fully utilize the existing cores and memory space, matching graphs with different sizes of signal propagation. At this stage, the error rate of the TimeWarp algorithm is very low, as discussed in Section IV. However, even if an error happens and leads to a rollback, the simulation rolls back to the checkpoint from the last clique, thus the rollback distance is much shorter than the original TimeWarp algorithm. Next, we start from the offline stage graph modeling and analysis in ZhouBi.

### A. *Offline Graph Analysis*

This section describes the transformation of a netlist into a task graph. We expand the content from two aspects: graph representation, skewness and perfectness.

**The Graph Representation.** ZhouBi digests the gate-level netlist from the output of RTL synthesis. The marco and cell in the netlist is automatically convert into a task vertex set $V$, while the connectivity info is transferred into a relation edge set $E$. Each node in the vertex set is called a task node, representing the function and location of one specific gate. The graph structure does record message passing direction info. However, in this first part of offline analysis, we reserve this direction. In order to find a suitable set to partition the graph $G$, based on the observations above, we compute the skewness and perfectness as the graph feature.

**Skewness and Perfectness.** The graph skewness $\mu(G)$ is defined as the sum of all minimum numbers of edges of connected components whose removal results in a planar graph [12]. Though the graph cardinality grows with the size of input signals, this skewness stays. Converting a plain graph to a planar one, the process is the same as finding maximum cliques in the plain graph, as known as computing the chromatic number of the target graph [22]. We use an improved coloring algorithm [23] to find all cliques with $V_{\text{clique}} \geq 3$. The graph skewness $\mu(G)$ is defined as

$$\mu(G) = \frac{\sum_{i=3}^{n} iN_{c_i}}{|V|} \tag{1}$$

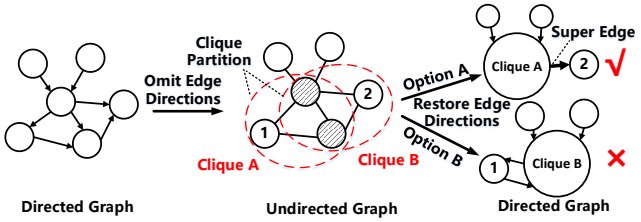


Fig. 5. Clique partition in ZhouBi.

where $N_{c_i}$ represents the number of i-clique. Note that, this process is offline, clique exploration would not affect the performance of the recursive simulation online. The perfect graph is a graph whose the chromatic number of every induced subgraph equals the clique number. The graph perfectness $p(G)$ is defined as

$$p(G) = \frac{\omega(G)}{\chi(G)} \tag{2}$$

where $\omega(G)$ represents the clique number of $G$, and $\chi(G)$ represents the chromatic number of $G$. We use $p(G)$ to estimate the size of possible conservative algorithm runs, which serves the worst case theoretical resource estimation.

### B. *Partition and Scheduling*

Based on the skewness $\mu(G)$ and perfectness $p(G)$, we start to partition the target graph, with the number of cliques found in the offline analysis.

**Clique Partition.** For each clique identified at the offline stage, we encapsulate the clique as a merged node in the graph, thus shrinking the cardinality of to-be computed graph. The maximum clique problem is finding the complete graph with the most vertices in an undirected graph. If two cliques intersect, we let the larger clique keep the overlapping nodes, and reduce the other clique into a smaller size. When the two intersecting cliques are the same size, it is necessary to judge the relationship between adjacent nodes. As shown in Figure 5, the encapsulation option with fewer communication times is adopted, i.e., Option A. During this process, we encapsulate small cliques, which appears to be a new node in the graph. These new nodes sufficiently fit into a thread thus we could process them parallelly.

**Cone Partition.** Another important partition is to prepare the critical path for the optimistic TimeWarp algorithm. We use the classic cone partition algorithm [24] to divide the graph from a set of nodes from the same level. First the primary input nodes are stored in a list. Traverse the list and perform a depth-first traversal on each primary input node in the list, and each node traversed retains a label of the primary input node to indicate which unit it belongs to. Thereby dividing the task graph into multiple cone-shaped partitions. In most cases, there inevitably are overlapping nodes between adjacent partitions. The overlapping parts between cones are organized by comparing by a ratio $\rho = \frac{s}{u}$, where the $s$ is the number of nodes and the $u$ is the total number of inputs and outputs. The larger the $\rho$ of the cone unit is, the better. Each cone partition serves independently in the later parallel computation.

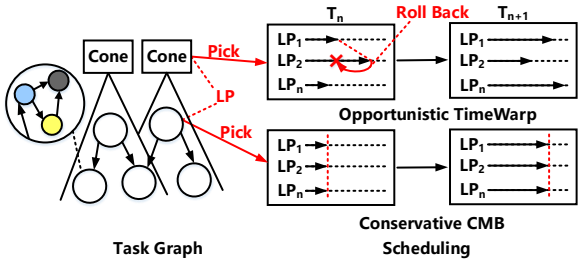


Fig. 6. Task scheduling in ZhouBi.

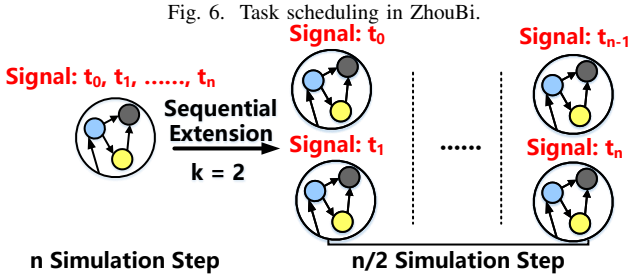


Fig. 7. Signal propagation in unlooping.

**Scheduling.** This component is based on the partial order relationship of the nodes in the graph to determine the priority of the node simulation sequence in the task graph. As described in the previous section, the strongly dependent part is encapsulated into clique, and the relationship between the newly generated clique and the remaining nodes is weakly dependent. However, the simulation sequence and parallelizability of nodes are still fuzzy. Therefore, we need to set the simulation priority of the clique and the simulation priority of the logic gate inside the clique. Nodes without precursors are used as primary input nodes and also as the first layer of simulation priority layer. And their successor nodes form the second level priority queue. And so on, the final result is a two-dimensional priority queue. In the simulation process, the two-dimensional priority queue advances the simulation process layer by layer, and the nodes at the same level of priority can be simulated in parallel. Based on all results from preprocessing above, the ZhouBi software defines logic process to perform simulation in a direct mapping. Each node in a cone may contain one clique or single node. All nodes at the same level from different cones can be processed simultaneously. We perform a CMB algorithm for simulation in nodes. As introduced later, our CMB algorithm is extended to map nodes from the same level to the suitable CPU core and associated memory chunk, thus the performance is further improved in a fine-grained manner. The simulation between nodes in the different cone adopts the opportunistic TimeWarp algorithm. This TimeWarp algorithm starts only when all nodes in the same cone have completed their local simulation after shuffling. The computation between neighbor cones are processed independently as they have a weak dependence in-between. Cones far apart can be directly mapped into cores, running individually. Such scheduling is simple and effective, as shown in Figure 6.

### C. Online Simulation in Parallel

**Cost Estimation in Propagation.** To further improve the simulation performance, ZhouBi unloops the input signal sequence along the time to make the size of tasks loaded on each core in the many-core system adapt to the current computing force, as is shown in Figure 7. For example, when the unlooping frequency $k$ is 2, the timing signals requiring $n$ steps can be completed in $n/2$ steps. ZhouBi could estimate the footprint of the clique $m$ based Equation (3).

$$m = \frac{c \times k \times n \times p \times 1\text{KB}}{P(G)} \ (1 \le k \le a) \tag{3}$$

where $n$ is the cardinality of the clique, $p$ is the number of input pins, $k$ is the frequency of unlooping, $a$ is the index number of testbench received by the clique, and $c$ is a user-defined coefficient, $P(G)$ is graph perfectness. We use $P(G)$ to estimate the total resource demand for each simulation, and $k$ to maximize each chunk of resource assigned.

**Simulation inside Cone Units in Parallel.** The nodes with strong internal data dependence formed after clique partition are contained in the cone unit. Besides, the number of nodes in the cone unit is smaller than that in the whole graph, so the granularity of node synchronization in the cone is smaller. We improve the original algorithm in a thread-level parallel optimization and combine with sequential extension inside clique to propose an optimized CMB-P algorithm. As shown in Algorithm 1 (Line 8-Line 12), iterate over the clique list and the simulation progress is promoted layer by layer according to clique's priority. We omit core information from the kernel, store it in a chain, and assign them to each

**Algorithm 1:** The ZhouBi Simulation Algorithm

**Input** : Cone unit list $uList$;
Unlooping Frequency $k$;
**Output** : Simulation results list $rList$;
**Parameter:** Cone unit $u$; Clique $c$; Current time $C_i$; New message $e$; Message time $t_e$; Max time $T$; Gate $g$; Input signal of gate $i_g$;

```
1  foreach u do                                   /* TW-P Map */
2      sequentialExtend(k);
3      while C_i < T do
4          revNewMessage()→ t_e;
5          if t_e < C_i and result is wrong then
6              rollBackTo(t_e) → C_i;
7          while clique layer is not None do
8              foreach g with same priority in c do
                   /* CMB-P Threads activate */
9                  collectInput(i_g);
10                 g.updateSignal(i_g);
11                 g.evaluate();        /* CMB-P Join */
12                 if c needs to send e to another u then
13                     add output to send queue;
14         foreach e do                 /* TW-P Shuffle */
15             sendNewMessage(e);
16 gatherResults()→ rList;              /* TW-P Reduce */
```

individual step, mainly in three stages: (1) Assign cores to simultaneously collect the external input received by nodes with the same priority (Line 9); (2) Place the input signal on the corresponding pin concurrently (Line 10); (3) Perform simulation at the same layer (Line 11). Compared to the CMB, the CMB-P, on the basis of the strong consistency of the simulation results, is improved in two aspects: 1) The simulation inside clique is divided into three more fine-grained steps, each step can be executed in parallel, which improves the parallelism of the simulation in the spatial domain; 2) The internal structure of the clique is extended to increase the throughput of the signals, which improves the parallelism of simulation in the time domain.

**Simulation between Cone Units in Parallel.** We improve the original TimeWarp algorithm by combining with the priority queue and get the TimeWarp-P (TW-P) algorithm, as shown in Algorithm 1. We use the TW-P algorithm to opportunistically travel each shortest path in the cone partition for the performance improvement. We parallelize this process in a Map Reduce manner. When all nodes are computed using the CMB-P algorithm above, each cone is mapped into a process and shuffled during the processing until ZhouBi detects the required communication between cones. Recursively, the TW-P algorithm in each cone is reduced one single output. All output files are merged in the later stage while nodes and edges are combined. The process level parallel is performed in these three stages: (1) Check new messages arrival and proceed the simulation along the path layer with the simulation priority queue (Line 4-Line 6); (2) Fetch the simulation inside the clique (Line 9-Line 13); (3) Message communication in a shuffle (Line 14-Line 15). Compared to TimeWarp, the TimeWarp-P, on the basis of the inherent parallelism guarantee, is improved in two aspects: 1) Combining with the priority queue, it further improves the efficiency of the cone path exploration and improves the data concurrency; 2) While coarse-grained synchronization, it provides the opportunity to combine with fine-grained synchronization algorithm to reduce the frequency of rollbacks to some extent.

**Time Complexity Analysis.** Compared to traditional simulation algorithms as mentioned in II-B, ZhouBi transforms the microtasks, represented by individual gates, into the macrotasks, represented by cliques. This approach reduces the dimensionality of the entire simulation and occurs in two aspects: 1) Structurally, where individual gates in the circuit are packed into cliques; 2) Temporally, where the timing signals of the whole simulation are unlooped and expanded on cliques, allowing multiple timing signals to be simulated in a single simulation step. As a result, ZhouBi's time complexity becomes $O(\frac{N_t}{k} \times (\frac{N_{c_1}}{1} + \ldots + \frac{N_{c_i}}{i}))$, where $N_t$ is the number of excitation signal frames, $k$ represents the unlooping frequency, $N_{c_i}$ represents the number of i-clique.

## IV. Evaluation

In this section, we prototype ZhouBi and evaluate its performance on classic benchmarks.

TABLE I
PERFORMANCE COMPARISON ON BENCHMARKS [13], [20], [21].

| Benchmark | # Nodes | # Cones | # Nodes-MC | # SK | # PF | # S2V | # S2I |
|---|---|---|---|---|---|---|---|
| s1494 | 690 | 10 | 216 | 0.00 | 0.40 | 24.2× | 22.8× |
| s13207 | 8651 | 62 | 2489 | 0.07 | 0.75 | 23.9× | 21.6× |
| s35932 | 17828 | 35 | 3823 | 0.30 | 0.75 | 20.6× | 18.7× |
| multiplier | 27318 | 128 | 5891 | 0.25 | 0.60 | 20.3× | 18.5× |
| decoder | 33000 | 2000 | 10047 | 0.91 | 1.00 | 18.4× | 16.7× |
| log2 | 32124 | 32 | 6652 | 0.17 | 0.43 | 26.0× | 24.4× |
| leon3mp | 545836 | 148 | 72154 | 0.14 | 0.43 | 27.6× | 25.9× |
| leon2 | 780456 | 126 | 92614 | 0.03 | 0.38 | 25.7× | 23.5× |
| leon3-avnet-3s1500 | 899632 | 109 | 104890 | 0.22 | 0.43 | 25.8× | 24.2× |

**# Nodes:** node number **# Cones:** cone number **# Nodes-MC:** node number in the max cone **# SK:** graph skewness **# PF:** graph perfectness **# S2V:** speedup compared to Veriwell **# S2I:** speedup compared to Iverilog

### A. Empirical Setup

We have prototyped ZhouBi as a gate-level simulator and evaluate it with benchmarks described in Section II. We compare ZhouBi performance with some state-of-the-practice (SOTP) benchmarks, such as Iverilog [1] and Veriwell [2]. We adapt a machine with 24 Intel Xeon 6126 cores at 2.60 GHz and a 256 GB RAM (CentOS v7.6). Using the hyper-threading technology [25], we virtualize 48 logical cores from 24 physical cores to improve CPU execution efficiency. We compile all programs using GNU GCC-6.5.0 with C++11 standards -std=c++11 and optimization flags -O2 enabled. In the specific implementation of parallel simulation, we use multi-thread to conduct internal simulation inside cones by thread library based on C++11 standard and multi-process to conduct simulation between cones by using MPI Framework OpenMPI 3.1.6 [26]. The following simulation time does not include the preprocessing time for the dissociation graph analysis of the benchmarks.

### B. Performance Comparison

Table I lists the benchmark statistics and the overall performance comparison between ZhouBi with two classic Verilog simulators. We measure the simulation time of completing ten thousand iterations of the testbench on the benchmarks. We run our own gate simulator using the maximum hardware concurrency of 48 logical cores on our platform. The average CPU utilization of all benchmarks is above 95%. ZhouBi is faster than Veriwell and Iverilog across all benchmarks, which is shown as Table I. The three largest speedup values we observed are 27.6× compared to Veriwell and 25.9× compared to Iverilog on leon3mp, 26.0× and 24.4× on log2, and 25.8× and 24.2× on leon3-avnet-3s1500.

### C. Overall Performance Statistics and Discussion

To better evaluate ZhouBi, we test the performance of each module individually as well as the performance of modules stacked on top of each other. The simulation results by using 6 different simulation baselines, specifically including the serial simulation (Serial), the CMB simulation (CMB), the TimeWarp simulation (TW), improved CMB-P simulation (CMB-P), improved TimeWarp-P simulation (TW-P), complete simulation algorithm (ZhouBi). The overall performance of the benchmarks is shown in Figure 8. The performance of the improved algorithm is many times higher than that of the original algorithm.

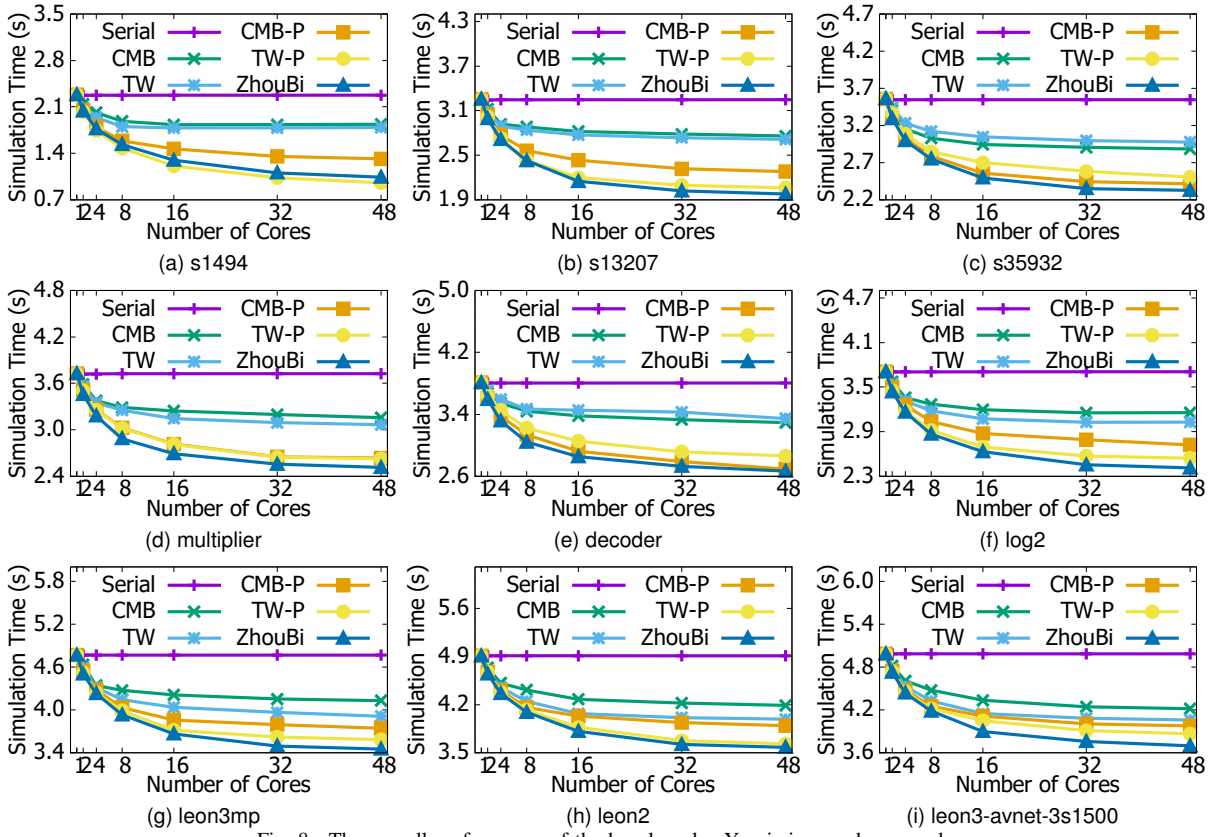


Fig. 8. The overall performance of the benchmarks. Y axis is on a $\log_{10}$ scale.

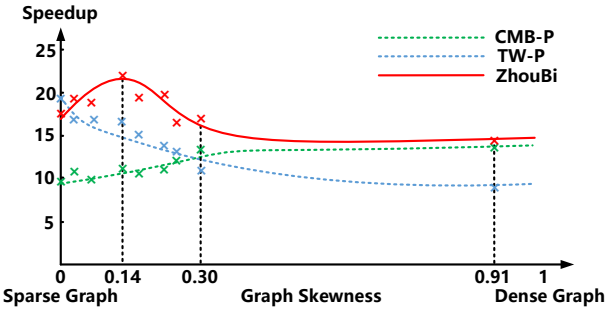


Fig. 9. The ZhouBi performance at different skewnesses.

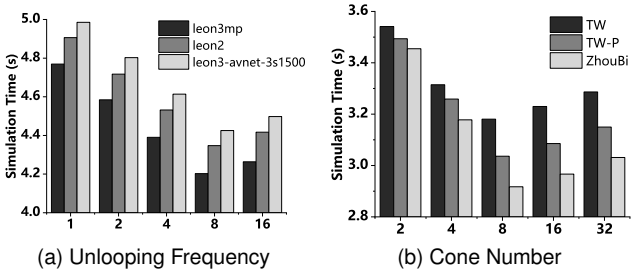


Fig. 10. Performance impact of unlooping frequency and cone number. Y axis is on a $\log_{10}$ scale.

**Impact of Graph Skewness on Performance.** The graph skewness is the key factor affecting the simulation performance, which is shown in Figure 9. The results show that (1) CMB-P algorithm is good at handling fine-grained tasks with strong data dependencies. As the graph becomes denser, there are more cliques in the graph. Benefiting from the parallel optimization brought by the sequential extension inside the clique, the performance gradually improves, but it still cannot fully utilize the inherent parallelism of simulation. (2) The TW-P opportunistic algorithm is good at handling coarse-grained tasks with weak data dependencies. When the graph is sparse, the data dependencies in the graph are weak, and the probability of rolling back is very low. However, as the graph becomes denser, the causal constraints in the graph increase, resulting in a higher probability of rolling back. A lot of failures of rolling back can cause a lot of extra overhead, resulting in performance degradation. When the skewness reaches a certain degree, for example, skewness reaches 0.3, the simulation performance is even worse than that of CMB-P conservative algorithm. (3) ZhouBi weighs the merits of CMB-P versus TW-P, scheduling CMB-P for fine-grained tasks with strong data dependence and TW-P for coarse-grained tasks with weak data dependence. In most of graphs with uneven skewness, ZhouBi outperforms CMB-P and TW-P because ZhouBi has both the performance improvement of TW's high concurrency and the advantage of conservative algorithm to reduce the rollback rate. In dense graph decoder, i.e. the skewness of the circuit is 0.91, the performance of ZhouBi converges to the CMB-P.

**Impact of Unlooping Frequency on Performance.** The unlooping frequency determines the number of signal simulation within a single simulation step of the clique, which is used to increase the parallelism of signal simulation in time sequence. We allocate different unlooping frequency to each clique of 3 industrial circuits to explore its impact on performance. The result is shown in Figure 10a. As the

unlooping frequency per clique increases, the simulation time of each case decreases first and then increases. When the frequency of unlooping is low, the throughput of input signal per clique in a single simulation step is small, which results in the insufficient utilization of the computing power of the core. As the frequency of unlooping increases, the throughput of input signal per clique gradually matches the computing power supported by the underlying many-core system, and the performance gradually improves. When the frequency of unlooping continues to increase and the throughput of input signal per clique exceeds the range of the computing power per core, the simulation performance declines.

**Impact of Load Balance between Cone Units on Performance.** The load size in the cone unit, i.e., the number of nodes in cone unit, is an important factor affecting the simulation performance between cones. We test on the *log2*. The circuit is partitioned according to the primary input number and 32 cone units are obtained. Then, adjust the load balance between cone units by combining the adjacent cone units. The result is shown in Figure 10b. With the increase of the number of final cone units, the simulation time decreases first and then increases. The minimum simulation time is reached when the number of cones is 8. When the number of the cone units is small, the concurrency between cone units is low and the simulation performance is weak. When the number of the cone units increases excessively, the communication overhead between cones increases, which leads to some noticeable performance degradation. Load balance between cone units is required to achieve better performance.

## V. Related Work

ZhouBi is a framework designed for improving the simulation performance by scheduling tasks into many-core systems under the premise of graph theory-guided netlist partition. Related works with similar features to ours are as follows.

**Netlist Partition.** The quality of netlist partition directly affects the simulation performance. Zheng et al. [27] propose a multi-level topology-driven partitioning framework for FPGA systems to deal with topological constraints in the system. Scarabotlo et al. [28] propose a new partitioning algorithm for gate-level error determination. The core idea of the algorithm is to partition according to the impact of each gate on the final output. These heuristic partitioning algorithms are difficult to trade off between partition communication and load balancing. In recent years, partition methods based on deep learning have also emerged. Lu et al. [29] propose a partitioning framework based on graph neural network (GNN). However, GNN-based methods need to design too many parameters, resulting in high computational complexity.

**Many-core System.** The operation of circuit simulation is strongly data-dependent and cannot be directly mapped to multi-core hardware to fully utilize computing resources. In order to improve the concurrency of data during simulation, Zeng et al. [30] propose a method for logic re-simulation on multi-core hardware via gate/event parallelism and state compression, enabling 2D parallelism by grouping gates and splitting events. Lai et al. [31] propose a new hybrid contention-tolerant parallel logic simulation algorithm to fully utilize the parallel capability of hardware. However, these parallel optimization schemes are all based on conservative strategies, ignoring the optimization opportunities brought by the opportunistic strategies.

**Hybrid Synchronization.** Conservative and opportunistic strategies have their own advantages. Eker et al. [32] propose a simulation synchronization scheme, which can dynamically switch between conservative and opportunistic strategies according to the features of the runtime. The essence of the studies is to choose a suitable strategy of conservative or opportunistic, without taking advantage of the respective benefits of both at the same time. ZhouBi uses the graph skewness and graph perfectness to schedule conservative and opportunistic strategies, and make full use of their respective benefits.

In all, we find some effective heuristics to provide the leverage between the conservative and opportunistic simulation. One interesting observation is that, large n-cliques ($n \geq 4$) are rare in these benchmarks. We can use this heuristic in the algorithm for finding cliques to improve the efficiency. This 3-clique dominates in the gate-level graph calls for some unique optimization design in the underlying support. However, our ZhouBi still has the following limitations: 1) The circuits that ZhouBi can support for acceleration are primarily large-scale circuits that are neither very dense nor very sparse. 2) Every time the circuit structure is changed, ZhouBi needs to conduct a incremental analysis of the graph structure. The performance of the graph algorithm also has a great impact on the performance of ZhouBi.

## VI. Conclusion

In this paper, we identify the performance problem of gate-level simulation with a focus on the graph representation. We systematically study many classic circuit graphs, and observe some unique features that can accelerate the simulation. We propose ZhouBi framework that provides a fast and scalable gate-level simulation. ZhouBi is fast because we optimize the gate-level simulation from their graph structure to a fine-grained architecture parallel support. ZhouBi supports scalability because ZhouBi picks the graph skewness to partition the graph. This skewness metric stays when the graph scales with the signal propagation, thus the partition stays with similar features. Our ZhouBi prototype can achieve a maximum 27.6× performance gain on modern benchmarks, as compared to the state-of-the-practice baselines.

## Acknowledgment

We thank our shepherd, Professor Xun Jiao, Professor Guangyu Sun, and other anonymous reviewers for their insightful comments. We also thank SMIT Group Limited for their support and in-depth discussions. This work is supported by the National Key R&D Program of China, No.2022YFB4501703, the Provincial Key Research and Development Program of Jiangxi (20212BBE53004), and the Major Key Project of PCL (PCL2022A05).

## REFERENCES


[1] S. Williams and M. Baxter, "Icarus verilog: open-source verilog more than a year later," *Linux Journal*, vol. 2002, no. 99, p. 3, 2002.

[2] D. C. Hyde, "Bucknell handbook on verilog hdl," *Com puter Science Department, Bucknell University Lewis burg*, 1995.

[3] A. J. L. Martin, "Cadence design environment," *New Mexico State University, Tutorial paper*, p. 35, 2002.

[4] J. Balkind, M. McKeown, Y. Fu, T. Nguyen, Y. Zhou, A. Lavrov, M. Shahrad, A. Fuchs, S. Payne, X. Liang *et al.*, "Openpiton: An open source manycore research framework," *ACM SIGPLAN Notices*, vol. 51, no. 4, pp. 217–232, 2016.

[5] A. B. Kahng, J. Lienig, I. L. Markov, and J. Hu, *VLSI physical design: from graph partitioning to timing closure*. Springer, 2011, vol. 312.

[6] W. Su and C. L. Seitz, "Variants of the chandy-misra-bryant distributed discrete-event simulation algorithm," *California Institute of Technology*, 1988.

[7] S. Sabarathinam and A. Prasad, "Generalized synchronization in a conservative and nearly conservative systems of star network," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 28, no. 11, p. 113107, 2018.

[8] B. Wang, Y. Zhai, Z. Wang, H. Zhang, and D. Qing, "Enhanced null message algorithm for hybrid parallel simulation systems with large disparity in time step," in *2016 IEEE/ACM 20th International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*. IEEE, 2016, pp. 61–68.

[9] A. Pellegrini and F. Quaglia, "A fine-grain time-sharing time warp system," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 27, no. 2, pp. 1–25, 2017.

[10] X. Liu and P. Andelfinger, "Time warp on the gpu: Design and assessment," in *Proceedings of the 2017 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, 2017, pp. 109–120.

[11] D. Jefferson and R. Fujimoto, "A brief history of time warp," in *Advances in Modeling and Simulation: Seminal Research from 50 Years of Winter Simulation Conferences*. Springer, 2017, pp. 97–134.

[12] M. Bartels, H. Wei, and D. C. Mason, "Dtm generation from lidar data using skewness balancing," in *18th International Conference on Pattern Recognition (ICPR'06)*, vol. 1. IEEE, 2006, pp. 566–569.

[13] C. Albrecht, "Iwls 2005 benchmarks," in *International Workshop for Logic Synthesis (IWLS): http://www. iwls. org*, 2005.

[14] F. Dörfler, J. W. Simpson-Porco, and F. Bullo, "Electrical networks and algebraic graph theory: Models, properties, and applications," *Proceedings of the IEEE*, vol. 106, no. 5, pp. 977–1005, 2018.

[15] G. Zhang, H. He, and D. Katabi, "Circuit-gnn: Graph neural networks for distributed circuit design," in *International conference on machine learning*. PMLR, 2019, pp. 7364–7373.

[16] M. R. Rohanipoor, B. Ghavami, and M. Raji, "Improving combinational circuit reliability against multiple event transients via a partition and restructuring approach," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 5, pp. 1059–1072, 2019.

[17] X. Yin, B. Sedighi, M. Varga, M. Ercsey-Ravasz, Z. Toroczkai, and X. S. Hu, "Efficient analog circuits for boolean satisfiability," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 1, pp. 155–167, 2017.

[18] P. Vaziri and K. Vora, "Controlling memory footprint of stateful streaming graph processing." in *USENIX Annual Technical Conference*, 2021, pp. 269–283.

[19] R. Xiao and C. Chen, "Gate-level circuit reliability analysis: A survey," *VLSI Design*, vol. 2014, 2014.

[20] L. Amarú, P.-E. Gaillardon, and G. De Micheli, "The epfl combinational benchmark suite," in *Proceedings of the 24th International Workshop on Logic & Synthesis (IWLS)*, 2015.

[21] N. Prasad, I. Chakrabarti, and S. Chattopadhyay, "An energy-efficient network-on-chip-based reconfigurable viterbi decoder architecture," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 10, pp. 3543–3554, 2018.

[22] T. Storch, "How randomized search heuristics find maximum cliques in planar graphs," *ACM*, 2006.

[23] A. H. Gebremedhin, D. Nguyen, M. M. A. Patwary, and A. Pothen, "Colpack: Software for graph coloring and related problems in scientific computing," *ACM Transactions on Mathematical Software*, vol. 40, no. 1, 2013.

[24] G. Saucier, D. Brasen, and J. Hiol, "Partitioning with cone structures," in *Proceedings of 1993 International Conference on Computer Aided Design (ICCAD)*. IEEE, 1993, pp. 236–239.

[25] D. T. Marr, F. Binns, D. L. Hill, G. Hinton, D. A. Koufaty, J. A. Miller, and M. Upton, "Hyper-threading technology architecture and microarchitecture." *Intel Technology Journal*, vol. 6, no. 1, 2002.

[26] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine *et al.*, "Open mpi: Goals, concept, and design of a next generation mpi implementation," in *European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting*. Springer, 2004, pp. 97–104.

[27] D. Zheng, X. Zang, and M. D. Wong, "Topopart: a multi-level topology-driven partitioning framework for multi-fpga systems," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2021, pp. 1–8.

[28] I. Scarabottolo, G. Ansaloni, G. A. Constantinides, and L. Pozzi, "Partition and propagate: An error derivation algorithm for the design of approximate circuits," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[29] Y.-C. Lu, S. Pentapati, L. Zhu, G. Murali, K. Samadi, and S. K. Lim, "A machine learning-powered tier partitioning methodology for monolithic 3-d ics," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 11, pp. 4575–4586, 2021.

[30] C. Zeng, F. Yang, and X. Zeng, "Accelerate logic re-simulation on gpu via gate/event parallelism and state compression," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2021, pp. 1–8.

[31] L. Lai, Q. Zhang, H. Tsai, and W.-T. Cheng, "Gpu-based hybrid parallel logic simulation for scan patterns," in *2020 IEEE International Test Conference in Asia (ITC-Asia)*. IEEE, 2020, pp. 118–123.

[32] A. Eker, Y. Arafa, A.-H. A. Badawy, N. Santhi, S. Eidenbenz, and D. Ponomarev, "Load-aware dynamic time synchronization in parallel discrete event simulation," in *Proceedings of the 2021 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, 2021, pp. 95–105.