SimPart: A Simple Yet Effective Replication-aided Partitioning Algorithm for Logic Simulation on GPU

31st International European Conference on Parallel and Distributed Computing (Euro-Par)

Yi-Hua Chung*[1], Shui Jiang*[2], Wan-Luan Lee[1], Yanqing Zhang[3], Haoxing Ren[3], Tsung-Yi Ho[2], and Tsung-Wei Huang[1]

University of Wisconsin at Madison[1], The Chinese University of Hong Kong[2], Nvidia Corporation[3]

August 29, 2025

Takeaways

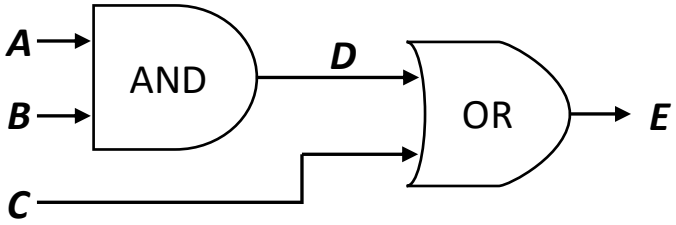
- Understand the fundamentals of logic simulation and partitioning
- Discuss the pros and cons of existing partitioning methods
- Present SimPart to overcome the limitations of existing partitioners
- Evaluate the performance of SimPart on industrial designs
- Conclude the talk

Takeaways

- **Understand the fundamentals of logic simulation and partitioning**
- **Discuss the pros and cons of existing partitioning methods**
- Present SimPart to overcome the limitations of existing partitioners
- Evaluate the performance of SimPart on industrial designs
- Conclude the talk

Logic Simulation

- Logic simulation is a key step for verifying the functionality of a digital design.
- A circuit is modeled as a directed graph, where nodes represent logic elements and edges represent data dependencies.
- Simulation evaluates this graph by propagating inputs through logic elements to produce outputs, often iterated over multiple testbenches for coverage.



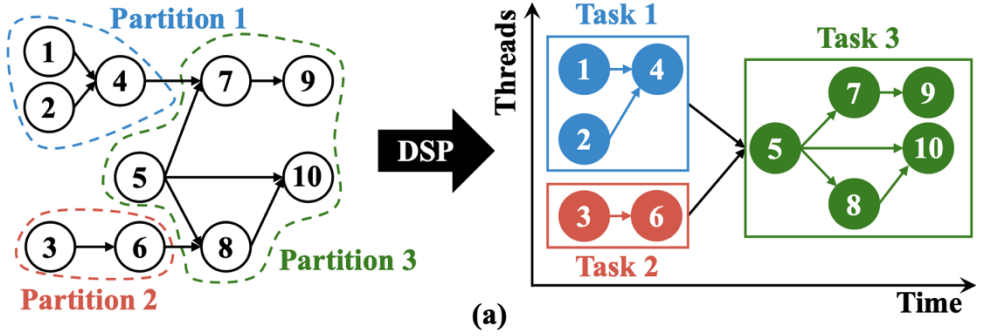
Pattern	Input			Internal	Output
	A	*B*	*C*	*D*	*E*
P_1	0	1	0	0	0
P_2	1	1	0	1	1
P_3	1	1	1	1	1

- Large designs can yield graphs with millions of nodes and edges, leading to long simulation times.

Graph Partitioning

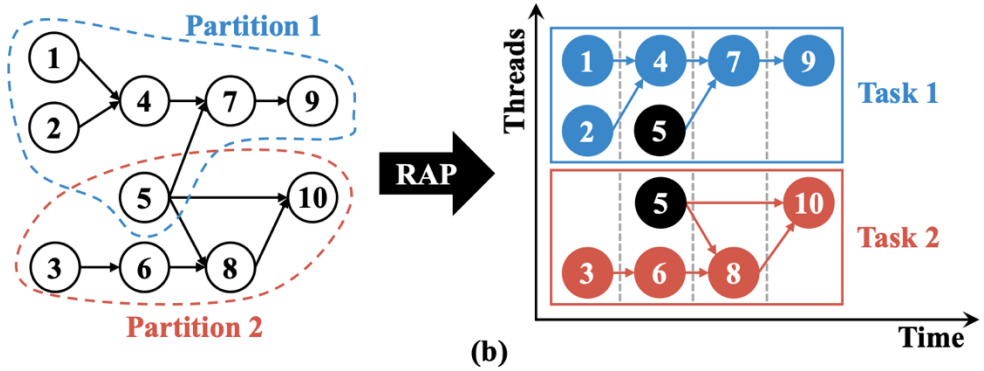
- To reduce simulation time, existing simulators improve parallelism by partitioning circuit graphs into a task dependency graph (TDG), where each task is a disjoint set of logic operations for parallel execution.

Disjoint set-based partitioning (DSP)



Method 1. DSP partitions the circuit graph to three non-overlapped tasks

Replication-aided partitioning (RAP)



Method 2. RAP divides the circuit into two independent tasks, with node 5 replicated in both

Pros and Cons of DSP and RAP

Method	Disjoint set-based partitioning (DSP)	Replication-aided partitioning (RAP)
✅ **Pros**	• Simple and widely used in simulators[13, 9, 11, 1] • Lower implementation complexity • Lower preprocessing overhead	• Reduces cross-partition dependencies • Improves task-level parallelism in circuits with moderate interconnections • Lower synchronization overhead than DSP
⚠ **Cons**	• Higher synchronization overhead when #levels is large • Cross-partition dependencies may limit parallelism • Load balancing overhead	• Over-replication can outweigh speedup benefits • Higher setup cost (proxy hypergraph + NP-hard partitioning[15]) • Higher memory usage due to replicated nodes

- DSP is easy and lightweight but may not scale well with deep circuit graph.
- RAP can unlock more parallelism but comes with higher setup cost and may face the over-replication problem.

13. Snyder, W.: Verilator 4.0: open simulation goes multi- threaded (2018)
9. Lin, D.L., et al.: From rtl to cuda: A gpu acceleration flow for rtl simulation with batch stimulus. In: Proceedings of the 51st ICPP. pp. 1–12 (2022)
11. Lin, D.L., et al.: TaroRTL: Accelerating RTL Simulation using Coroutine-based Heterogeneous Task Graph Scheduling. In: Euro-Par (2024)
1. Beamer, S., et al.: Efficiently Exploiting Low Activity Factors to Accelerate RTL Simulation. In: ACM/IEEE DAC. pp. 1–6 (2020)
15. Wang, H., et al.: RepCut: Superlinear Parallel RTL Simulation with Replication-Aided Partitioning. In: ACM ASPLOS. p. 572–585 (2023)

Takeaways

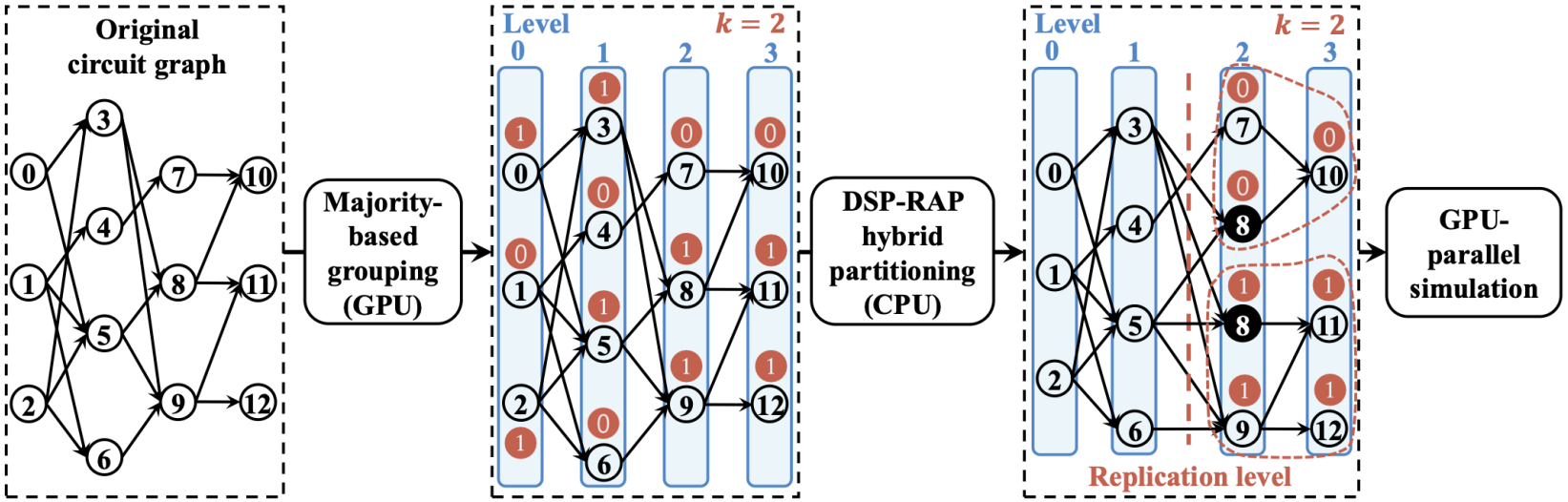
- Understand the fundamentals of logic simulation and partitioning
- Discuss the pros and cons of existing partitioning methods
- **Present SimPart to overcome the limitations of existing partitioners**
- Evaluate the performance of SimPart on industrial designs
- Conclude the talk

SimPart

- To fully leverage the strengths of DSP and RAP, we present **SimPart**, a GPU-parallel, replication-aided partitioner that balances both methods and speeds up logic simulation on a GPU.
- In this work:
  1. We propose a simple yet effective algorithm that directly solves the partitioning problem without solving another time-consuming proxy problem.
  2. We leverage the circuit graph property to design a DSP-RAP hybrid strategy that limits replications and balances parallel efficiency.
  3. We adopt conditional CUDA Graphs to build a GPU-parallel simulation that cuts kernel-call and control-flow overhead.

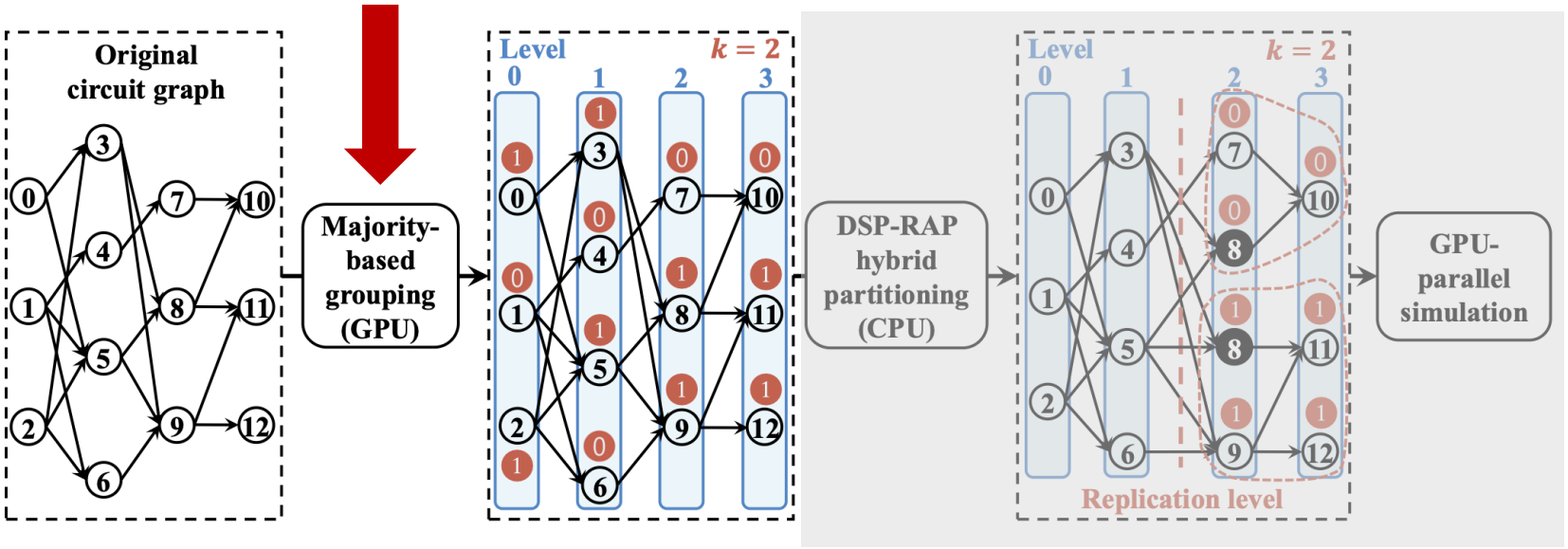
SimPart Overview

- This figure gives an overview of **SimPart**, which consists of three stages:
  1. Majority-based grouping
  2. DSP-RAP hybrid partitioning
  3. GPU-parallel simulation



SimPart Overview

- This figure gives an overview of **SimPart**, which consists of three stages:
  1. **Majority-based grouping**
  2. DSP-RAP hybrid partitioning
  3. GPU-parallel simulation

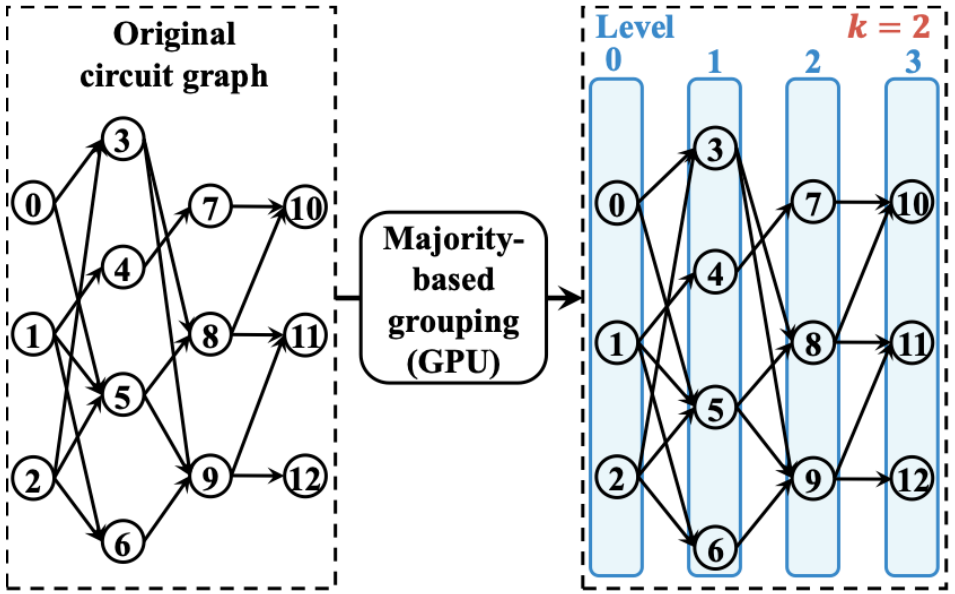


Step 1 - Majority-based Grouping

- **Goal**: The goal of majority-based grouping is to determine a partition group ID for each node while minimizing the number of replications, using a one-pass BFS on the GPU.
- **Steps**:
  - Step 1.1 - Initial grouping
  - Step 1.2 - Group propagation

Step 1.1 - Initial Grouping

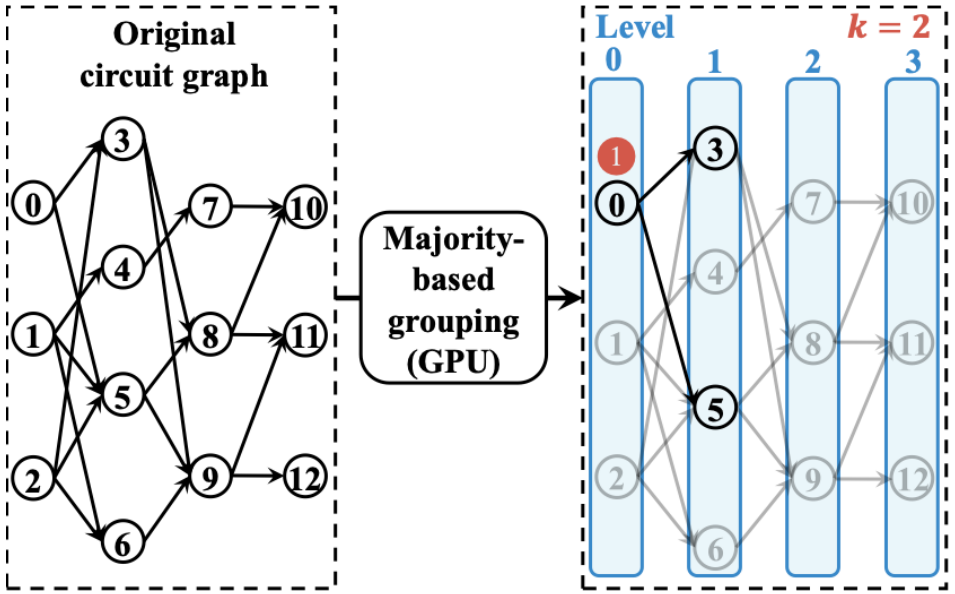
- **Goal**: Assign each input node a group ID based on the majority count of its successors, modulo the number of partitions (k).
  - The reason for using the majority count is to encourage the later propagation phase to place connected nodes in the same group.



Partition size (k) = 2

Step 1.1 - Initial Grouping

- **Goal**: Assign each input node a group ID based on the majority count of its successors, modulo the number of partitions (k).
  - The reason for using the majority count is to encourage the later propagation phase to place connected nodes in the same group.



Partition size (k) = 2

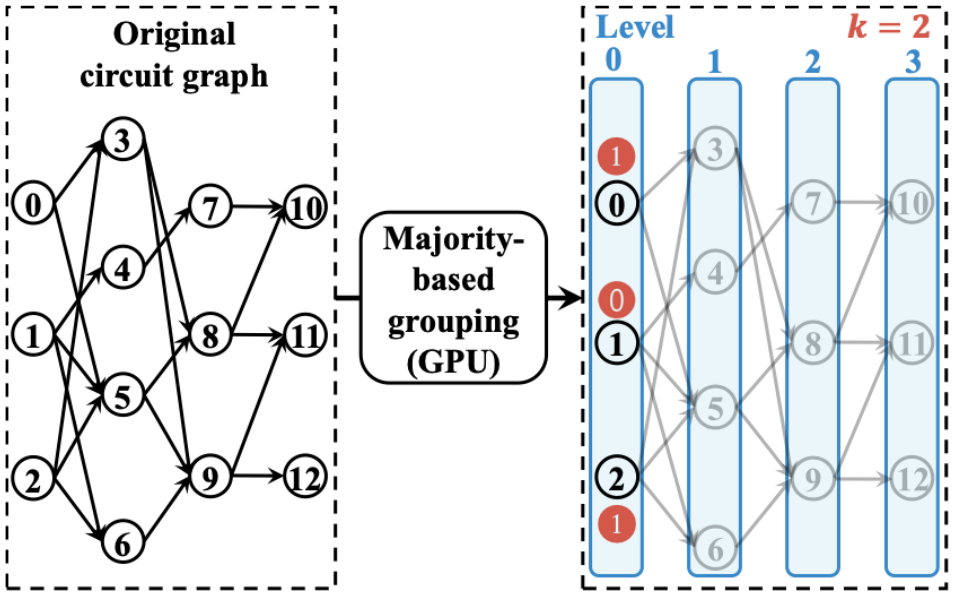
- Node 0 has two successors: node 3 and 5.
- Here's how they map to groups:
  - 3 mod 2 = 1 ⟶ **Group 1**
  - 5 mod 2 = 1 ⟶ **Group 1**

Group ID	Accumulated Count
Group 0	0
Group 1	2

- Since Group 1 has the highest count, we place node 0 in **Group 1**.

Step 1.1 - Initial Grouping

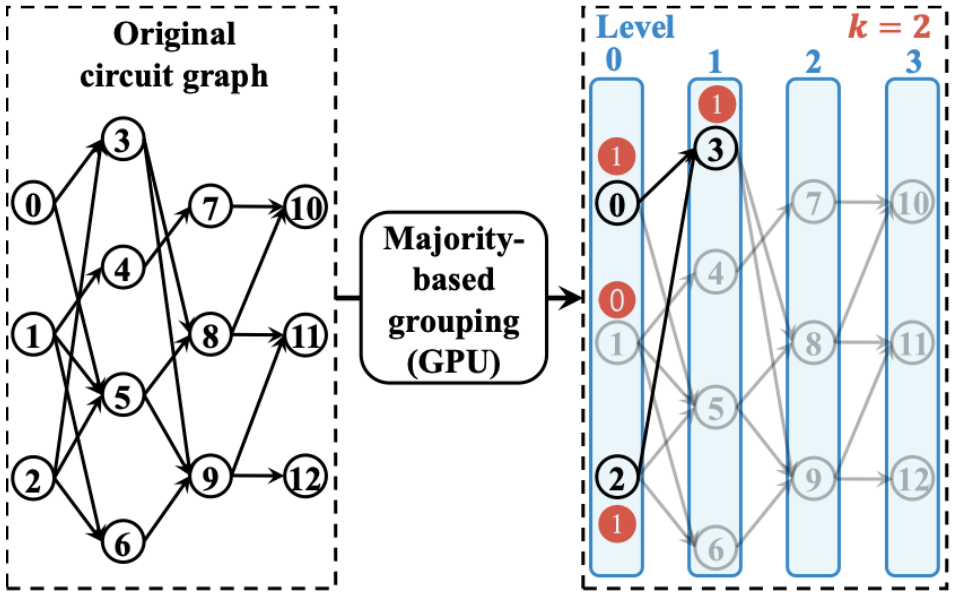
- **Goal**: Assign each input node a group ID based on the majority count of its successors, modulo the number of partitions (k).
  - The reason for using the majority count is to encourage the later propagation phase to place connected nodes in the same group.



Partition size (k) = 2

Step 1.2 - Group Propagation

- **Goal**: Propagate these group IDs from inputs to outputs, keeping connected nodes in the same group as much as possible.



Partition size (k) = 2

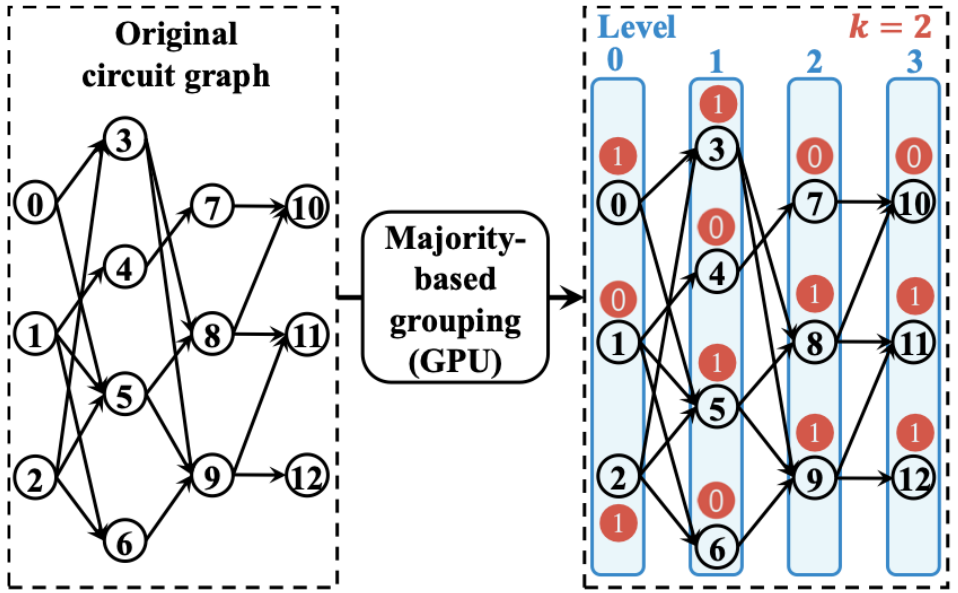
- Node 3 has two predecessors: node 0 and 2.
- Here's how they're grouped:
  - Nodes 0 and 2 are in **Group 1**

Group ID	Accumulated Count
Group 0	0
Group 1	2

- Since Group 1 has the highest count, we assign node 3 to **Group 1**.

Step 1 - Majority-based Grouping

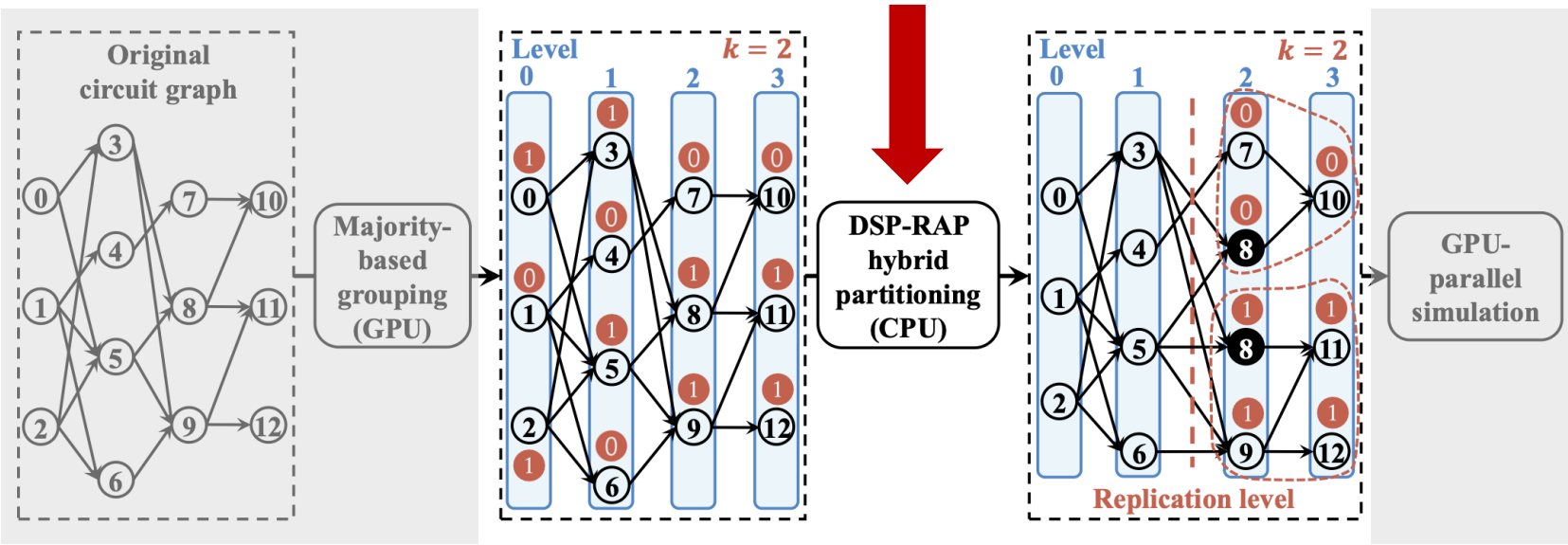
- **Goal**: Propagate these group IDs from inputs to outputs, keeping connected nodes in the same group as much as possible.
- We perform a BFS to traverse the circuit graph level by level on GPU.



Partition size (k) = 2

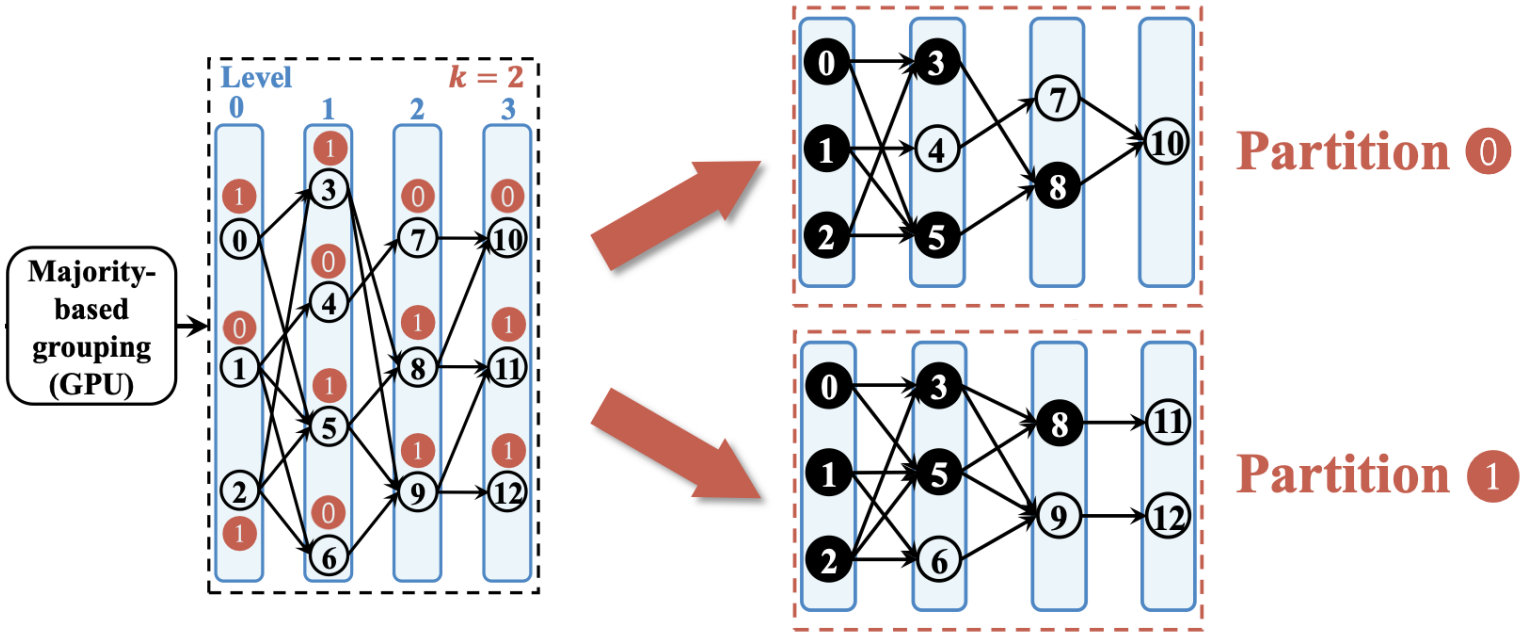
SimPart Overview

- This figure gives an overview of **SimPart**, which consists of three stages:
  1. Majority-based grouping
  2. **DSP-RAP hybrid partitioning**
  3. GPU-parallel simulation

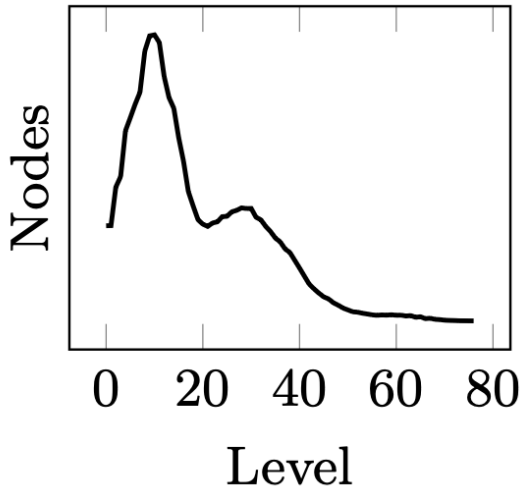


Problem and Circuit Property

- **Problem**: Using pure RAP often leads to **over-replication**, especially when backward traversals from outputs go too deep toward level 0.
- **Property**: Circuit graphs often show a long-tailed distribution after levelization, with more parallel nodes at earlier levels.



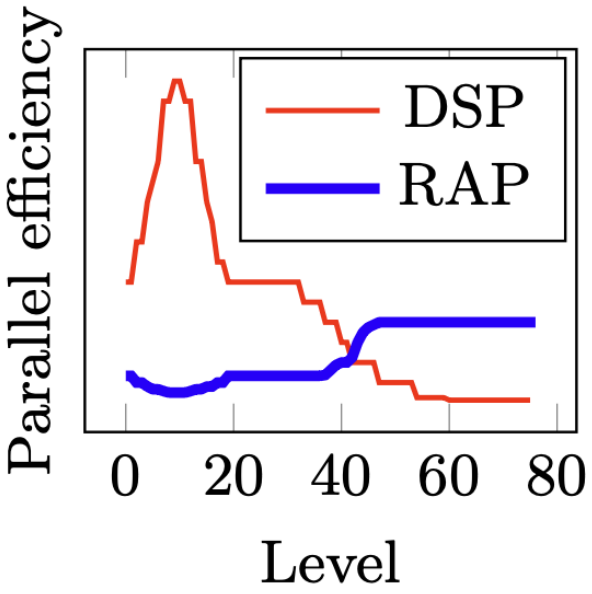
Problem: Over-replication



Property: Long-tailed distribution

DSP vs RAP

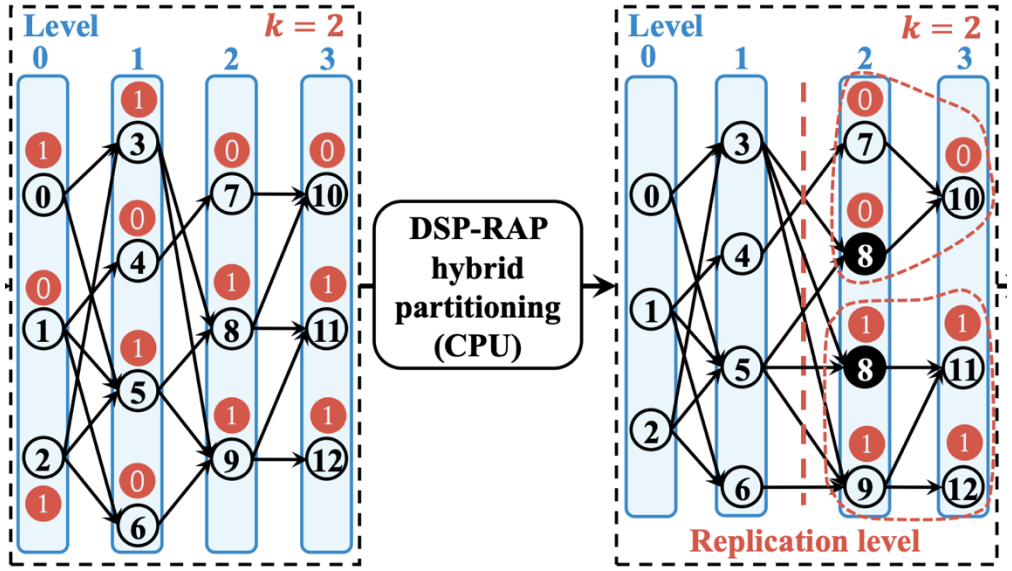
- To illustrate the impact, we roughly characterize the **parallel efficiency** as the ratio of GPU thread utilization to synchronization overhead.
- We compare two partitioning extremes on the same circuit and GPU.
  - **DSP**: run level-by-level simulation on the GPU with the highest parallel efficiency
  - **RAP**: replicate overlaps and create k independent tasks for simulation



- **Early levels: DSP** achieves high efficiency by simulating all parallel nodes at once, maximizing thread usage. However, **RAP** suffers from over-replication, resulting in poor efficiency.
- **Later levels: DSP**'s efficiency declines as it must synchronize all thread blocks after each level (inter-block sync). **RAP**, however, only needs to synchronize threads within the same block (intra-block sync), maintaining higher efficiency at deeper levels.

Step 2 - DSP-RAP Hybrid Partitioning

- To balance parallel efficiency, we propose a DSP-RAP hybrid strategy that **limits replication up to a threshold level** during backward traversal.
- Let γ be the upper bound on the number of parallel nodes. Replication stops at the first level whose width exceeds γ; we call this the **replication level**.



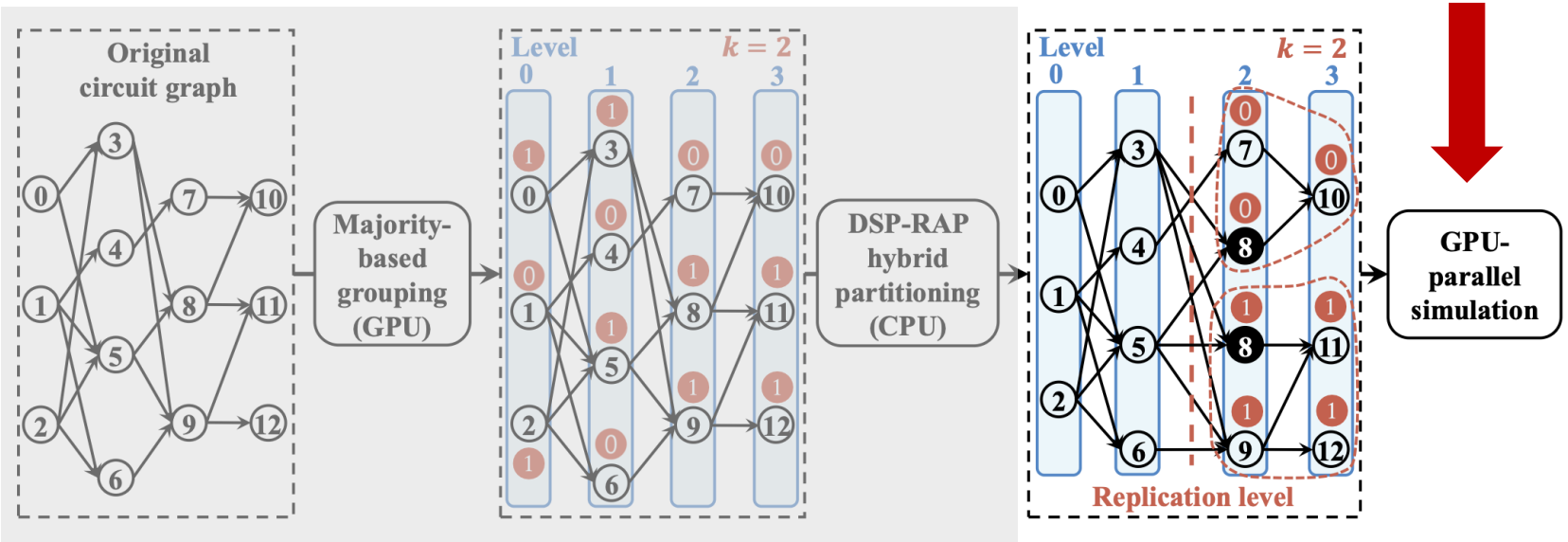
Partition size (k) = 2

- When γ = 3, the replication level is level 1, the first level with more than three parallel nodes.
- **For levels before the replication level**, we use **DSP** for simulation.
- **For levels after the replication level**, we switch to **RAP** for simulation.
- In the DSP-RAP hybrid partitioning, we define γ to cap replication, which lets us **combine the strengths of DSP and RAP**.

Parallel efficiency: *We roughly characterize the parallel efficiency to the ratio of GPU thread utilization to synchronization overhead.*

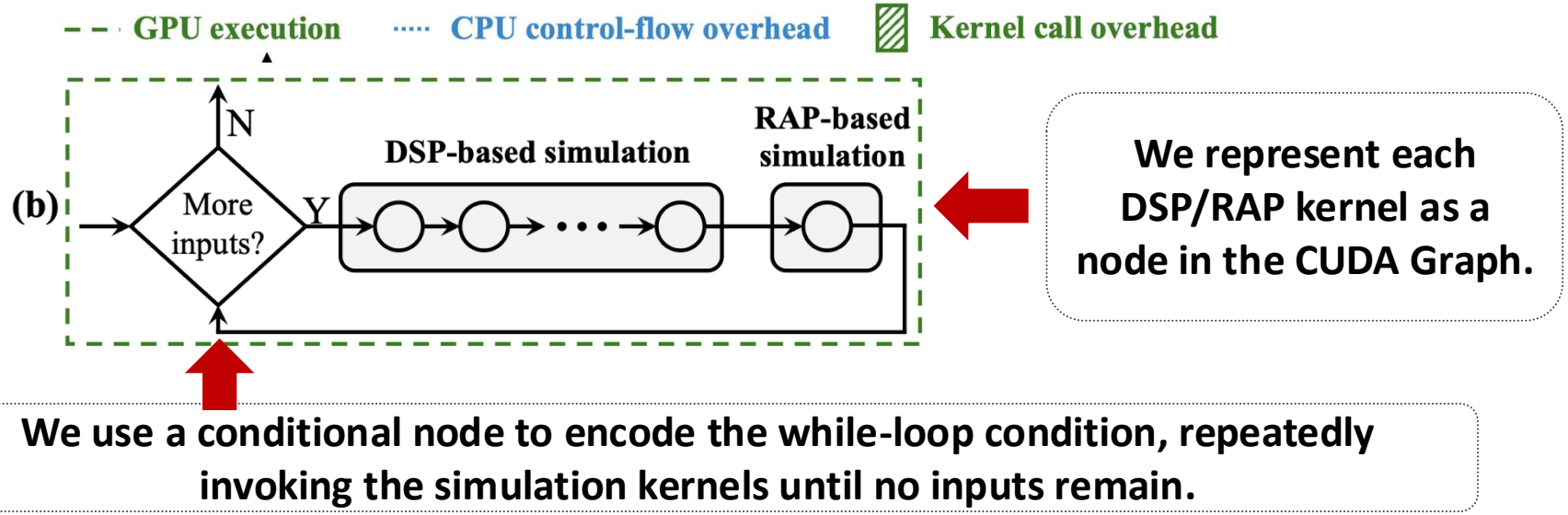
SimPart Overview

- This figure gives an overview of **SimPart**, which consists of three stages:
  1. Majority-based grouping
  2. DSP-RAP hybrid partitioning
  3. **GPU-parallel simulation**



Step 3 - GPU-parallel Simulation

- Logic simulation on GPUs faces two issues, **GPU kernel call overhead** and **control-flow overhead**.
- To address these issues, we run the entire loop as a **Conditional CUDA Graph**[3], so both kernels and control decisions execute **on the GPU**.



- This significantly reduces kernel launch and control-flow overhead.

Takeaways

- Understand the fundamentals of logic simulation and partitioning
- Discuss the pros and cons of existing partitioning methods
- Present SimPart to overcome the limitations of existing partitioners
- **Evaluate the performance of SimPart on industrial designs**
- Conclude the talk

Experimental Setup

- We evaluated **SimPart** on logic graphs generated by the RTLFlow[9] simulator.
- We implemented SimPart in CUDA/C++ and compiled it with nvcc v12.3 using -O3 and -std=c++20.
- We consider the state-of-the-art **RepCut**[15] as our baseline.
- We ran the experiments on a 64-bit Linux machine with 32 Intel Core i5-13500 cores (4.8 GHz) and an Nvidia RTX A4000 GPU.
- All data is an average of 10 runs.

9. Lin, D.L., et al.: From rtl to cuda: A gpu acceleration flow for rtl simulation with batch stimulus. In: Proceedings of the 51st ICPP. pp. 1–12 (2022)
15. Wang, H., et al.: RepCut: Superlinear Parallel RTL Simulation with Replication-Aided Partitioning. In: ACM ASPLOS. p. 572–585 (2023)

Performance Evaluation - Partition

Benchmark	$\|V\|$	$\|E\|$	RepCut [15] T^{C1}_{Build}	T^{C1}_{Part}	T^{C16}_{Part}	T^{C1}_{Sim}	T^{C16}_{Sim}	T^{G}_{Sim}	$R_{\|V\|}$	$R_{\|E\|}$	SimPart T_{Part}	T^{G}_{Sim}	$R_{\|V\|}$	$R_{\|E\|}$
edit_dist	164K	164K	590.3	1492	438.6	439.5	281.8	5.1	1.01765	1.01765	72.3	3.2	1.00528	1.00528
matrix_mult	176K	174K	737.4	1024	384.6	895.6	450.2	6.2	1.00235	1.00235	86.6	3.4	1.03148	1.03148
b19	255K	255K	2058	4377	1203	169.1	40.3	7.5	1.00911	1.00911	77.3	4.0	1.00210	1.00210
leon2	1.6M	1.6M	10054	12478	5501	6846	900.2	35.5	1.00166	1.00166	213.3	21.2	1.00000	1.00000
leon3mp	1.2M	1.2M	6893	12542	4513	4648	528.5	27.3	1.00487	1.00487	146.1	16.1	1.00000	1.00000
netcard	1.5M	1.5M	9120	24578	7209	7589	1184	30.7	1.02060	1.02060	155.9	18.4	1.00000	1.00000
edit_dist_eval	1.3M	1.3M	5193	7432	3148	54791	10555	25.6	1.00209	1.00209	160.5	18.1	1.00000	1.00000
matrix_mult_eval	1.4M	1.4M	6531	5545	2886	76191	13465	29.0	1.00004	1.00004	779.2	17.3	1.00002	1.00002
b19_eval	2.0M	2.0M	15976	37299	10872	4157	811.6	39.4	1.00147	1.00147	925.2	24.3	1.00000	1.00000
leon2_eval	12.9M	12.9M	91500	64697	44711	165011	29598	235.5	1.00021	1.00021	2483	169.5	1.00000	1.00000
leon3_eval	20.0M	20.0M	106012	128188	57139	219415	38024	327.5	1.00004	1.00004	2971	259.0	1.00000	1.00000
netcard_eval	24.0M	24.0M	143731	766218	218582	1530950	283890	371.8	1.00000	1.00000	2866	292.4	1.00000	1.00000
Avg. ratio	**1.000**	**1.000**	-	**68.45**	**23.13**	**1344.76**	**255.04**	**1.58**	**1.005**	**1.005**	**1.000**	**1.000**	**1.003**	**1.003**

$|V|$: Number of nodes

T_{Build}: Hypergraph construction time

T^{Cn}_{Sim}: Simulation time on CPU, with n threads

T_{Part}: Partitioning time

$|E|$: Number of edges

T^{Cn}_{Part}: Partitioning time on CPU, with n threads

T^{G}_{Sim}: Simulation time on GPU

$R_{|V|}$, $R_{|E|}$: Ratio of replicated nodes and edges

15. Wang, H., et al.: RepCut: Superlinear Parallel RTL Simulation with Replication-Aided Partitioning. In: ACM ASPLOS. p. 572–585 (2023)

Performance Evaluation - Simulation

Benchmark	$\|V\|$	$\|E\|$	RepCut [15]								SimPart			
			T^{C1}_{Build}	T^{C1}_{Part}	T^{C16}_{Part}	T^{C1}_{Sim}	T^{C16}_{Sim}	T^{G}_{Sim}	$R_{\|V\|}$	$R_{\|E\|}$	T_{Part}	T^{G}_{Sim}	$R_{\|V\|}$	$R_{\|E\|}$
edit_dist	164K	164K	590.3	1492	438.6	439.5	281.8	5.1	1.01765	1.01765	72.3	3.2	1.00528	1.00528
matrix_mult	176K	174K	737.4	1024	384.6	895.6	450.2	6.2	1.00235	1.00235	86.6	3.4	1.03148	1.03148
b19	255K	255K	2058	4377	1203	169.1	40.3	7.5	1.00911	1.00911	77.3	4.0	1.00210	1.00210
leon2	1.6M	1.6M	10054	12478	5501	6846	900.2	35.5	1.00166	1.00166	213.3	21.2	1.00000	1.00000
leon3mp	1.2M	1.2M	6893	12542	4513	4648	528.5	27.3	1.00487	1.00487	146.1	16.1	1.00000	1.00000
netcard	1.5M	1.5M	9120	24578	7209	7589	1184	30.7	1.02060	1.02060	155.9	18.4	1.00000	1.00000
edit_dist_eval	1.3M	1.3M	5193	7432	3148	54791	10555	25.6	1.00209	1.00209	160.5	18.1	1.00000	1.00000
matrix_mult_eval	1.4M	1.4M	6531	5545	2886	76191	13465	29.0	1.00004	1.00004	779.2	17.3	1.00002	1.00002
b19_eval	2.0M	2.0M	15976	37299	10872	4157	811.6	39.4	1.00147	1.00147	925.2	24.3	1.00000	1.00000
leon2_eval	12.9M	12.9M	91500	64697	44711	165011	29598	235.5	1.00021	1.00021	2483	169.5	1.00000	1.00000
leon3_eval	20.0M	20.0M	106012	128188	57139	219415	38024	327.5	1.00004	1.00004	2971	259.0	1.00000	1.00000
netcard_eval	24.0M	24.0M	143731	766218	218582	1530950	283890	371.8	1.00000	1.00000	2866	292.4	1.00000	1.00000
Avg. ratio	**1.000**	**1.000**	-	**68.45**	**23.13**	**1344.76**	**255.04**	**1.58**	**1.005**	**1.005**	**1.000**	**1.000**	**1.003**	**1.003**

$|V|$: Number of nodes

T^{Cn}_{Sim}: Simulation time on CPU, with n threads

$|E|$: Number of edges

T^{G}_{Sim}: Simulation time on GPU

T_{Build}: Hypergraph construction time

T_{Part}: Partitioning time

T^{Cn}_{Part}: Partitioning time on CPU, with n threads

$R_{|V|}$, $R_{|E|}$: Ratio of replicated nodes and edges

15. Wang, H., et al.: RepCut: Superlinear Parallel RTL Simulation with Replication-Aided Partitioning. In: ACM ASPLOS. p. 572–585 (2023)

Takeaways

- Understood the fundamentals of logic simulation and partitioning
- Discussed the pros and cons of existing partitioning methods
- Presented SimPart to overcome the limitations of existing partitioners
- Evaluated the performance of SimPart on industrial designs
- Conclude the talk

Thank you for your time, and I’m happy to take any questions.

Email: yihua.chung@wisc.edu

Personal blog: