LEGOSim: A Unified Parallel Simulation Framework for Multi-chiplet Heterogeneous Integration

MICRO 2025 Submission #**NaN** – Confidential Draft – Do NOT Distribute!!

Abstract

The rise of multi-chiplet integration challenges existing simulators like gem5 [45] and GPGPU-Sim [37] for efficiently simulating heterogeneous multiple-chiplet systems due to incapability to modularly integrate heterogenous chiplets, high synchronization overheads in parallel simulation, and high inter-chiplet communication modeling overhead. To address these limitations, this paper introduces LEGOSim, a unified parallel simulation framework capable of flexibly integrating various open-source and in-house designed chiplet simulators as processes in parallel simulation, referred to as "simlets" with minimal modifications needed. It introduces a three-stage simulation process that decouples chiplet simulation from inter-chiplet communication modeling to mitigate the communication modeling overhead. The framework also integrates Network-on-Interposer (NoI) simulator for modeling inter-chiplet communication, enabling accurate assessment of various interconnection architectures' performance. Furthermore, it employs an on-demand synchronization protocol, ensuring synchronization only occurs when necessary, thus reducing overhead while maintaining correctness. Evaluated with diverse benchmarks, LEGOSim shows high accuracy in simulating multi-chiplet architectures like SIMBA [55] and a CiM-based accelerator [13], with average errors of 3.79% and 3.94%, respectively. It significantly reduces synchronization overhead by up to 99.9% compared to per-cycle synchronization and by 66.1% compared to time quantum synchronization, without synchronization errors. Five case studies show that LEGOSim also provides precise system performance metrics and stall cause reporting, simplifying tasks such as performance analysis and optimization, and can be used for design space exploration of various multi-chiplet systems.

Keywords

Simulator, architectural simulation, multi-chiplet system simulation.

1 Introduction

As semiconductor technology approaches its physical limits, multi-chiplet integration has become an essential design paradigm for the post-Moore era. Compared to traditional monolithic chip architectures, multi-chiplet systems package multiple heterogeneous chiplets (such as CPUs, GPUs, NPUs, CiMs, etc.) into a single system, which not only enhances computational performance but also optimizes power consumption, reduces costs, and improves chip yield. However, these highly integrated architectures also bring unprecedented challenges in design space exploration, especially in terms of the system-level simulation and evaluation.

MICRO 2025, October 18–22, 2025, Seoul, Korea
2025. ACM ISBN 978-X-XXXX-XXXX-X/XX/XX
https://doi.org/XXXXXXX.XXXXXXX

The challenges of architectural level multi-chiplet system simulation include:

1. Lack of modular integration flexibility: Numerous simulators have been developed to simulate individual components/chiplets such as CPUs, GPUs, and NoIs [2]- [66], as shown in Table 1. While these simulators are highly detailed and accurate for their specific targets, they lack the flexibility to be integrated into multi-chiplet systems as they are not designed for modular integration.

2. Synchronization inefficiency: To overcome the slow simulation speed problem of sequential simulation [22], parallel simulation with per-cycle synchronization [9] and time quantum synchronization [9] were proposed. However, per-cycle synchronization incurs huge synchronization overhead, while time quantum improves speed by relaxing the synchronization to be performed for each time quantum and but degrades accuracy.

3. Overhead in modeling inter-chiplet communication: Inter-chiplet communication modeling faces a trade-off between accuracy and performance. For instance, gem5 incurs substantial communication modeling overhead as the communication event handling is performed sequentially. On the other hand, simplified approaches compromise accuracy. For instance, Sniper uses queuing theory based analytical model to replace detailed network modeling, and trace based NoI-only simulation [11] ignores data dependency, both of which lead to high errors when the system scale increases.

To address these challenges, we propose LEGOSim, a unified parallel simulation framework for heterogeneous multi-chiplet systems, which is released in [6]. In LEGOSim, (1) the seamless integration of various chiplet simulators as modular simulator processes (referred to as simlets) such as gem5, sniper, GPGPU-Sim, etc. is enabled to flexibly model various multi-chiplet systems by parallel simulation for the design space exploration purpose, (2) an on-demand synchronization scheme is proposed to minimize synchronization overhead in parallel simulation while keeping accuracy, and (3) a simlet simulation and inter-chiplet communication modeling decoupled simulation strategy is proposed to reduce inter-chiplet communication modeling overhead.

The accuracy of LEGOSim has been validated with two published works, SIMBA [55] and a compute-in-memory (CiM) based accelerator architecture [13]. The simulation errors are below 10%, confirming its fidelity.

LEGOSim is showcased by five case studies to help explore the design space in multi-chiplet system design flows, including identifying performance bottlenecks, and design space exploration for inter-chiplet interconnection network and buffer size, inter-chiplet network topology selection, memory interfaces, and inter-chiplet interconnection protocols, demonstrating the versatility of LEGOSim in multi-chiplet system design flows.

The contributions of this paper are as follows:

Table 1: Summation of existing simulators. (Note: AI Acc stands for AI accelerator and CiM stands for compute-in-memory.)

Simulator	Target	Simulator	Target	Simulator	Target	Simulator	Target
SimBricks [41]	CPU	gem5 [45]	GPU	ROCm [34]	GPU	SimpleSSD [30]	SSD
Sniper [25]	CPU	MacSim [35]	CPU	Arbitor [27]	AI Acc	SSDExplorer [66]	SSD
ZSim [53]	CPU	Manifold [60]	GPU	NeuroSim [17]	AI Acc	BookSim [29]	NoC
GPGPU-Sim [37]	CPU/GPU	MGPU-sim [57]	GPU	Scale-Sim [52]	AI Acc	Garnet [7]	NoC
Graphite [46]	CPU/GPU	Nsight Compute [40]	GPU	MNSIM 2.0 [65]	CiM	Noxim [16]	NoC
Multi2Sim [58]	CPU/GPU	Nsight Systems [40]	GPU	DRAMsim3 [42]	DRAM	Ns-3 [15]	NoC
Accel-Sim [32]	GPU	PPT-GPU [10]	GPU	Ramulator [36]	DRAM	OMNeT++ [59]	NoC
Beignet [26]	GPU	OpenVINO Toolkit [2]	GPU	MQSim-E [38]	SSD	**LEGOSim**	**CPU+GPU+NPU+...**

- We propose an **on-demand synchronization scheme** that triggers synchronization only during inter-chiplet communication, reducing overhead by 99.9% compared to per-cycle synchronization while preserving accuracy.
- We propose a **three-stage decoupled simulation strategy** that decouples chiplet simulation from inter-chiplet communication modeling, improving efficiency and accuracy.
- We propose a **Unified Integration Interface (UII)** to enable seamless integration of diverse simulators (e.g., gem5, Sniper, GPGPU-Sim) into LEGOSim with parallel simulation and minimal code changes.
- We **implemented and open-sourced LEGOSim** [6], integrating multiple simlets, and invite researchers to contribute to design space exploration for multi-chiplet systems with LEGOSim.

2 Background & Motivation

2.1 Limitations of Existing Simulators in Modular Integration

In recent years, multi-chiplet architectures have been widely adopted in high-performance computing (HPC) and AI chips due to their superior scalability and energy efficiency. Notable examples include AMD's Zen 5 [8] with modular CCD/IOD design, supporting 32-64 cores and delivering over 2 TFLOPS of computing power. However, the design space exploration for such systems remains highly challenging due to the vast configuration space and complex interdependencies across interconnects, memory hierarchies, and communication protocols. For instance, Intel's Ponte Vecchio [28] integrates 47 chiplets and over 100 billion transistors, with a design cycle of a few years [4].

The limitations of existing simulators—especially their inability to support modular integration and high synchronization overhead— exacerbates low efficiency in design space exploration. Numerous simulators have been developed to simulate individual components/chiplets such as CPUs, GPUs, and NoIs [2]- [66], as tabulated in Table 1, which unfortunately lack the flexibility to be integrated to simulate heterogenous multi-chiplet systems. Modular simulators aim to integrate various components into a unified framework. For example, SimBricks [41] can integrate multiple simulators, but its complex integration mechanism results in low simulation speed, and it cannot model inter-chiplet transmission. ZSim [53] can efficiently simulate large-scale systems, but it has accuracy issues in simulating multi-chiplet interconnection networks. In addition, gem5-X [49] and its extended series (e.g., gem5-GPU [48], gem5-AcceSys [44], gem5-SALAM [51], etc.) also attempt to provide integration of CPUs, GPUs, memory models, and accelerators. However, these integrations require deep internal modification of the simulators, and they are fixed architectures, instead of modular integration of many other system architectures. SST (Structural Simulation Toolkit [50]) is another modular framework that supports integration across different simulation models and allows component plug-ins. However, SST cannot model inter-chiplet communication network and has significant simulation overhead and complexity, and also needs significant code modification to existing simulators.

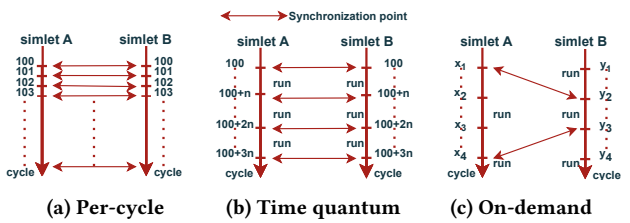


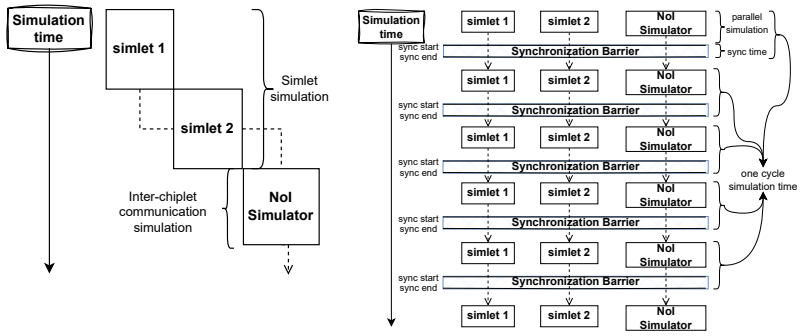
Figure 1: Comparison of different synchronization mechanisms. In Figure (c), x_i and y_i are the times at which inter-chiplet communication requests are initiated.

2.2 Limitations of Existing Simulator Synchronization Schemes

Synchronization overhead in parallel simulation remains a bottleneck to simulate a large-scale system with dozens of chiplets or a wafer scale computing system. Traditional synchronization methods, including sequential simulation [22], per-cycle synchronization [9], and time quantum synchronization [9] have following limitations.

1) Sequential simulation. In sequential simulation (e.g., gem5 [45]), the simulation of each simlet (i.e., the simulation module of an individual chiplet) and Network-on-Interposer (NoI) simulation is performed sequentially, resulting in low utilization of computing resources. Moreover, as the system size scales up, the simulation time grows exponentially. Figure 2a further illustrates the execution of sequential simulation, where the execution of simlets and NoI is strictly sequential with no overlap, significantly limiting the simulation efficiency. For example, with gem5 simulating one second of a many-core system takes 1 and 10 weeks [18], making sequential simulation impractical for large scale multi-chiplet systems.

2) Per-cycle synchronization. Parallel simulation improves efficiency compared to sequential simulation. Per-cycle synchronized parallel simulation (e.g., parti-gem5 [18]) allows the simulation of



(a) sequential simulation **(b) per-cycle simulation**

Figure 2: Simulation flow of sequential simulation and per-cycle synchronized parallel simulation.

multiple simlets to be executed in parallel and can overlap with the simulation of NoI, as shown in Figure 1a. However, synchronization is required in each simulation cycle, as shown in Figure 2b, leading to significant synchronization overhead. Per-cycle synchronization typically relies on shared memory, pipe communication, or file systems in host machines which not only increases the synchronization overhead but also causes serious performance bottlenecks, especially when simulating large-scale systems. As shown in Figure 3, the synchronization overhead increases drastically with the number of cores, making per-cycle synchronization infeasible for large scale systems.

3) Time quantum synchronization. To mitigate the synchronization overread in PC, Time Quantum (TQ), using a fixed time window (such as in Zsim [53]), was proposed to reduce the number of synchronizations, as shown in Figure 1b, but the size of the time window (x) needs to be manually adjusted. A large time window masks short-period cross-chiplet events (such as inter-chiplet data transmission or synchronization for the benchmark/application threads), leading to timing errors and low accuracy; on the other hand, a small time window degrades to near per-cycle synchronization, incurring huge simulation time.

As shown in Figure 3, we evaluate the above three synchronization mechanisms. The experiment is conducted with three different multi-/many-core configurations: 8 cores, 16 cores, and 32 cores,

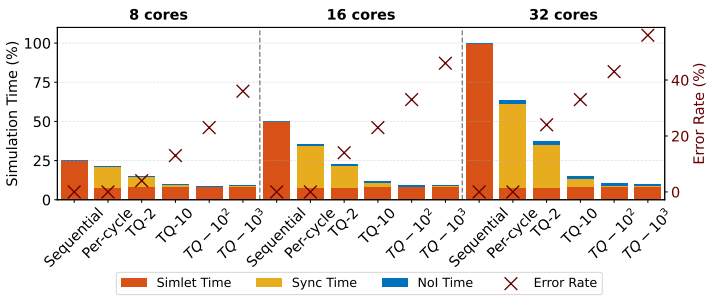


Figure 3: Comparison of overheads of different synchronization methods. Error is computed with respect to the sequential simulation.

using gem5 [45] for sequential simulation and parti-gem5 [18] for parallel simulation, modified to support both per-cycle synchronization and TQ-x modes (where TQ-x refers to the time quantum synchronization, i.e., synchronization per x cycles). As the number of cores increases, sequential simulation becomes highly inefficient and cannot scale to large systems. For the 16-core and 32-core configurations, the synchronization overhead of per-cycle synchronization increases rapidly. In particular, the synchronization time occupies a large percentage (85% in the 32-core case) of the total simulation time, significantly reducing overall simulation efficiency. In contrast, TQ synchronization improves efficiency by increasing the time window and reducing the frequency of synchronizations, but comes at the cost of increased synchronization errors. For example, with $TQ-10^3$ on the 32-core configuration, the synchronization overhead is reduced by 99.9% compared to per-cycle synchronization, but the error reaches 56%.

To address the above challenges, we propose **on-demand synchronization**. A key observation in multi-chiplet system design is that, inter-chiplet communication should be minimized by improving locality, as inter-chiplet communication latency is much higher than on-chip (intra-chiplet) interconnections and inter-chiplet communication bandwidth is much lower than on-chip (intra-chiplet) interconnections [61]. Inspired by this observation, on-demand synchronization performs synchronization only when communication occurs between chiplets, avoiding frequent synchronizations. This allows simlets to run independently without unnecessary interaction and stalls until inter-chiplet communications of other simlets occur, significantly reducing synchronization overhead, as shown in Figure 1c. This mechanism enables the system to maintain high simulation accuracy while substantially improving simulation efficiency.

Furthermore, inter-chiplet communication modeling poses a trade-off between accuracy and efficiency. Oversimplified communication models fail to model system-level performance accurately, while detailed modeling results in excessive simulation overhead, increasing simulation time to impractical levels for large-scale multi-chiplet systems. For instance, gem5 [45] incurs substantial communication overhead as all components (sim-objects) must communicate via port connections. Each communication is structured as a pair of request and response events, managed by a centralized

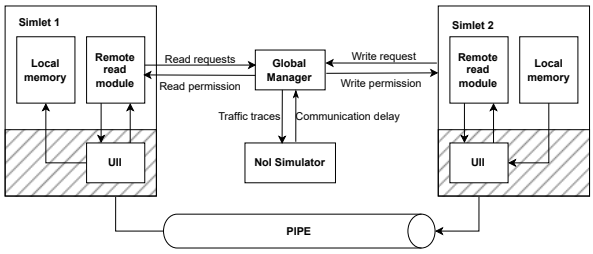


Figure 4: Overview of LEGOSim architecture and its components.

event queue and processed sequentially. As a result, simulation time escalates to days or weeks and in many cases the simulation crashes due to out of memory for large scale systems which hinders agile design space exploration [41]. In contrast, Sniper [25] reduces communication overhead by using queueing model based analytical modeling to replace detailed network transmission simulation, which unfortunately leads to degraded simulation accuracy as congestion cannot be faithfully modeled with correct timing. Similarly, performing the NoI (Network-on-Interposer) only simulation with traffic traces leads to low accuracy as data dependency is not considered [11]. To address this challenge, we propose a three-stage simulation mode, by **decoupling NoI modeling from chiplet simulation**, allowing NoI simulation to be performed separately after functional simulation, and integrating the inter-chiplet communication delay in the final stage with correct timing to ensure simulation accuracy. This method significantly reduces inter-chiplet communication modeling while maintaining accuracy and simulation efficiency.

3 LEGOSim Architecture and Design Principles

The design principles of LEGOSim are two key mechanisms. One is *on-demand synchronization* to improve simulation efficiency, which triggers synchronization only upon inter-chiplet communications, thereby significantly reducing synchronization overhead. The other is *decoupling inter-chiplet communication from chiplet simulation*, which optimizes performance and fidelity.

3.1 Overview of LEGOSim

LEGOSim supports parallel simulation and breaks down the simulation of a multi-chiplet system into the following three components, as shown in Figure 4:

1)**Heterogeneous Chiplet Simulation Units (Simlets)**: Different simlets (CPUs, GPUs, NPUs, CiMs, etc.) are independent simulation processes in parallel simulation, each of which can be existing open-source simulators (e.g., gem5 [45] or Sniper [25] for CPU chiplets, GPGPU-Sim [37] for GPU chiplets, MNSIM [65] for compute-in-memory chiplets, etc.). Simlets interact with each other through a Unified Integration Interface (UII), which will be described in Section 4.

2) **Network-on-Interposer (NoI) Simulator**: Used for modeling the interconnection topologies of inter-chiplet network, to accurately simulate inter-chiplet communication latency.

Table 2: Comparison of synchronization mechanisms

Sync Method	**Per-Cycle**	**Time-Quantum**	**LEGOSim (On-Demand)**
Frequency of Sync	Every cycle	Every n cycles	Only on inter-chiplet communication
Accuracy	High	Medium	High
Overhead	Very High	Medium	Very Low
Scalability	Poor	Moderate	Good
Notes	Precise but costly	Trade-off between speed and fidelity	Syncs only when necessary

3) **Global Manager (GM)**: Responsible for coordinating inter-chiplet data synchronization, scheduling NoI simulation, and executing synchronization strategies. The GM employs on-demand synchronization to minimize synchronization overhead while ensuring simulation accuracy.

Simlets perform their respective chiplet simulation in parallel, communicate and synchronize with the GM through the UII, while the GM coordinates these simlets' synchronization and data transfers, ensuring the accuracy of the simulation. The NoI simulator simulates the communication behavior between chiplets and provides the GM with communication delay of inter-chiplet data transfer, thus enabling the GM to make correct synchronization decisions.

3.2 Decoupling Inter-chiplet Communication Modelling from Chiplet Simulation

In traditional simulation, NoI simulation is interwoven with functional simulation. To improve the simulation efficiency, LEGOSim decouples NoI modeling, separating it from functional simulation, and integrates inter-chiplet communication delay in the final stage to ensure simulation timing accuracy. Figure 5 shows the simulation workflow of LEGOSim, including the following three stages.

1) Stage 1: LEGOSim simulates the functional model of each chiplet independently, while collecting inter-chiplet communication traffic traces. Each simlet executes functional simulation independently, advancing its local clock tick, ensuring that the functional behavior of each chiplet is accurately modeled. Meanwhile, all inter-chiplet communication events are recorded, including the time of data packet injection into the inter-chiplet network as traffic traces. They are crucial for the subsequent stage because they will be used to analyze inter-chiplet communication delays. Inter-chiplet communication traffic traces include timestamps of packet injection, data packet size, source and destination chiplets, etc. Since this stage does not perform any form of inter-simlet synchronization, each simlet performs its simulation without considering inter-chiplet communication delays or network congestion which incurs no stalls. This decoupled design maximizes parallel simulation efficiency and avoids additional synchronization overhead.

2) Stage 2: The inter-chiplet communication traffic traces collected in the first stage are input into the NoI simulator for inter-chiplet network interconnection modeling. In this stage, the NoI simulator runs independently, modeling inter-chiplet communications, generating accurate delay results for each traffic flow. As a comparison, conventional parallel simulation requires that each simlet and NoI should be strictly synchronized and advance cycle by cycle, and thus leads to huge synchronization and communication overhead.

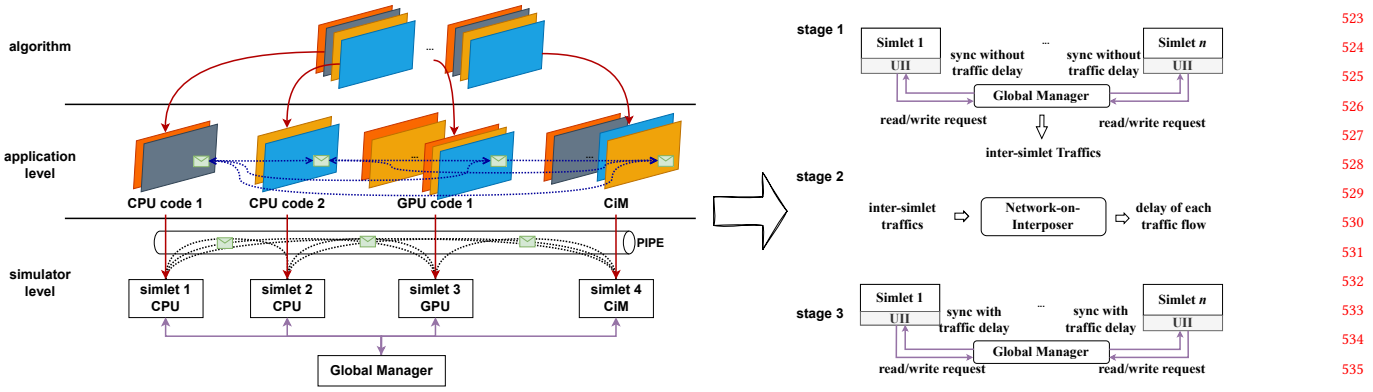


Figure 5: Workflow of the three-stage decoupled simulation of LEGOSim.

3) Stage 3: The system integrates the execution time of simlets and the inter-chiplet communication latency from the NoI simulation in the second stage into the system-level simulation. The GM uses these delay values to adjust the clock cycles of each simlet, ensuring that the timing of inter-chiplet communication is accurately modeled and the entire system simulation models the actual behavior of the multi-chiplet system. Simlets re-execute their functional and timing models, now incorporating inter-chiplet communication delays. The GM coordinates the synchronization of all simlets, ensuring that shared resource access and inter-chiplet communication delays are correctly aligned. By integrating inter-chiplet communication delays in the final stage, LEGOSim achieves high-fidelity simulation of the entire multi-chiplet system, accurately capturing the functional behavior and timing characteristics of inter-chiplet communications.

LEGOSim's decoupling simulation strategy effectively addresses the challenges of simulating complex multi-chiplet systems. By decoupling the functional simulation of individual chiplets from detailed modeling of inter-chiplet communication, LEGOSim achieves both accuracy and efficiency.

3.3 On-Demand Synchronization Mechanism

Traditional synchronization methods in parallel simulation often incur high computational overhead as shown in Table 2. Thus, LEGOSim adopts an on-demand synchronization, allowing simlets to run independently and synchronize only when inter-chiplet communication occurs.

As shown in Figure 6, inter-simlet synchronization is coordinated by the GM which is a controller thread/process. The workflow follows these steps:

① **Read/Write or Synchronization Request (Step 1)**: A simlet generates a read/write request as well as a clock synchronization request and sends it to the GM, which includes timing information, i.e. the simlet's local clock cycle, and this simlet stops advancing its clock ticking and waits for the response from the GM.

② **Global Manager Handling Requests (Step 2)**: For read/write requests, the GM pairs the data requester and responder between simlets, following a producer-consumer model for ordered inter-chiplet communication. For synchronization requests, the GM calculates the next target clock cycle to be advanced for the simlet, coordinating with other active simlets to maintain consistent timing across the system. In simulation stage 1, where inter-simlet traffic delays are not known yet, the GM sets the clock cycle to be advanced for the simlet to be the maximum cycle of the two communicating simlets. In simulation stage 3, in contrast, the GM takes into accounts NoI transmission latency from simulation stage 2. Here, the clock cycle to be advanced for the data-sending simlet remains the maximum clock cycle of the two simlets, while the data-receiving simlet's clock cycle to be advanced is adjusted by adding the corresponding NoI delay.

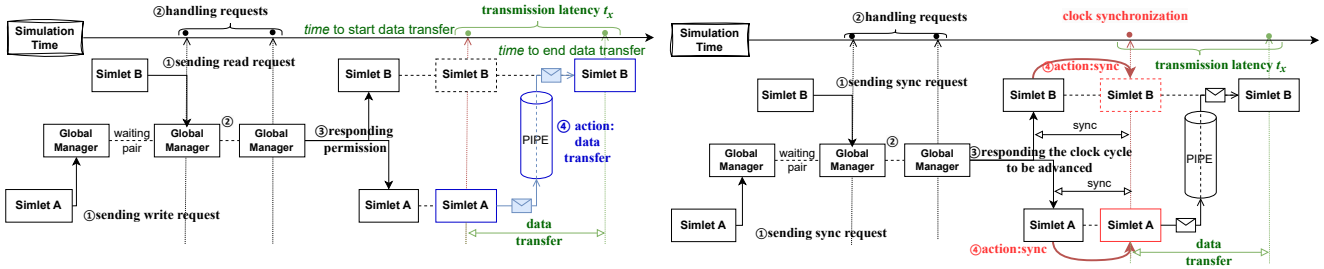
③ **Request Response (Step 3)**: After processing the request, the GM returns a response to the simlet. For read/write requests, this response indicates the readiness of the data transfer, allowing the simlet to proceed with the requested inter-chiplet read or write operation. For synchronization requests, this response specifies the clock cycle to be advanced.

④ **Data Transfer Execution/Clock Synchronization (Step 4)**: Upon receiving the synchronization response, the simlet advances its clock to the designated cycle. This step completes the clock synchronization, ensuring that all simlets currently remain aligned across the simulation. With synchronization confirmed, the simlet performs the data transfer operation. This step may involve reading or writing remote chiplet data according to the initial request.

By dynamically adjusting the synchronization points based on actual inter-chiplet data exchange, the system achieves both high simulation efficiency and accuracy.

4 Unified System Integration

The Unified Integration Interface (UII) is a fundamental component of the LEGOSim framework to support modular parallel simulation. It is designed to provide a standardized framework for integrating diverse simulators—whether they model CPUs, GPUs, DRAMs, or custom domain-specific accelerators (DSAs)— to simulate a multi-chiplet system in parallel. The UII abstracts simulator-specific interfaces and harmonizes them under a unified API, providing benchmark/application-level APIs, system call mapping, data



(a) Workflow of the inter-simlet data transfer. **(b) Workflow of the inter-simlet clock synchronization.**

Figure 6: Workflow of the inter-simlet synchronization.

transfer management, and clock synchronization mechanisms. Figure 7 outlines its modules, which include:

1) Benchmark/Application-Level APIs and System Call Definition: The UII defines a standard set of benchmark-level APIs used by chiplets for inter-chiplet communication and synchronization: *sendMessage*() and *receiveMessage*(). When integrating a new simlet, these APIs must be mapped to the internal mechanisms of the simulator as follows.

- For system-call based simulators (e.g., gem5 [45], Sniper [25]), these APIs are implemented as custom syscalls (e.g., *SYSCALL_REMOTE_READ* and *SYSCALL_REMOTE_WRITE*) and processed by the syscall handling routine.
- For runtime-library-based simulators (e.g., GPGPU-Sim [37]), these APIs are mapped to existing functions (e.g., *cudaMemcpy*()).
- For DSA simulators (e.g., Scale-sim [52]), these APIs are embedded as function calls or files within the simulation script.

2) Data Transfer Implementation: Data transfer between chiplets in the UII is managed by functions such as *sendSync*(), *receiveSync*(), *write_data*(), and *read_data*(). These functions coordinate data transfer protocols with the GM and enable data transmission through dedicated channels as follows.

- For CPU simulators (e.g., gem5 [45], Sniper [25]) , *sendMessage*() / *receiveMessage*() are translated to be inter-simlet data transmission in the syscall handling routines by file exchange, pipes, or shared memory in the host machine. Data is transferred from and to this simlet's internal simulated memory.
- For GPU simulators (e.g., GPGPU-Sim [37]), additional memory copy operations (e.g., *cudaMemcpy*()) are inserted before/after calling *sendSync*() and *receiveSync*() to move data between this simlet and others. These wrappers ensure that the GPU's memory space remains consistent with LEGOSim's global model.
- For DSA simulators (e.g., Scale-sim [52]), UII writes inputs to an interface file, executes the DSA script, then reads output. *sendMessage*() is implemented by writing input data to this file, or passing arguments to the Python configuration function for the simlet. *receiveMessage*() reads output data after simulation completes.

3) Clock Control: Given the diversity of simulation timing models, UII supports a flexible synchronization model to ensure that heterogeneous simlets advance their respective local clock tick correctly as follows.

- For cycle-accurate simulators (e.g., gem5 [45], GPGPU-Sim [37]), their clock cycles are controlled. For example, gem5 uses an event-driven model of clock tick granularity, and synchronization is managed by controlling *tick*. In GPGPU-Sim, simulation progress is tracked using *gpu_sim_cycle* and *gpu_tot_sim_cycle*. Clock ticking is controlled by these variables in such simulators.
- For non-cycle-driven simulators (e.g., Sniper [25]), UII inserts pseudo operations to artificially delay execution, such as *Sleep*() to adjust the clock delay according to the synchronization events.
- For DSA simulators (e.g., Scale-Sim [52]), which have no native clock or with simplified execution timeline: A block of operations/computations is performed to obtain the execution time, which is reported to the GM for synchronization.

Below are examples of how it facilitates integration in Sniper [25] , GPGPU-Sim [37] and Scale-Sim [52]:

Integration of Sniper [25]. Sniper, a CPU simulator, required additional adaptation due to its non-cycle-driven execution. Custom system calls are defined (*SYSCALL_REMOTE_READ* and *SYSCALL_REMOTE_WRITE*) to map Sniper's remote read/write operations to UII's *sendMessage*() and *receiveMessage*() functions. In the functional model, these system call handling routine translates *receiveSync*(), *read_data*(), *sendSync*(), and *write_data*() into inter-simlet message passing. In the timing model, *readSync*() and *writeSync*() are used for synchronization. However, since Sniper does not advance by discrete clock cycles, a *Sleep*() function is inserted to adjust its

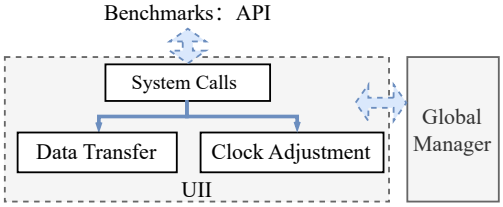


Figure 7: Modules of the UII.

Table 3: Configurations used in the experiments.

Configurations Used in the Simulation							
Sniper Configuration			**GPGPU-Sim Configuration**		**MNSIM Configuration**		
Cores	8 x86_64 ISA		# of SMs	80	Memristor Model	RRAM	
L1 D Cache	32KB, 8-way, 64B line, 4cycles, 1port		Tensor Core	640	Weight Bit	8	
L1 I Cache	32KB, 4-way, 64B line, 4cycles		Architecture	NVIDIA Volta (Titan V)	Crossbar Size	24	
L2 Cache	256KB, 8-way, 64B line, 8cycles, 1port		L1 Cache	32KB	**DSA**		
L3 Cache	8192KB, 16-way, 64B line, 30cycles, 4ports		L2 Cache	4.5MB	# of MACs	128	
memory size	2 GB				Global Buffer	64 KB (SRAM)	
Configurations of chiplet interposer							
Interposer					**Chiplet**		
Heat Thermal	Capacity	1.81×10^6 J/(m^3 · K)	Area	2500 mm^2	Chiplet Pitch	10 mm	
	Conductivity	35 /(m · K)	Thickness	0.1 mm	Capacitance Density	300 nF/mm^2	
Inter-chiplet Transmission Energy Consumption		1.17 PJ/bit					

execution timing according to the target clock cycles to be advanced, ensuring accurate synchronization.

Integration of GPGPU-Sim [37]. GPGPU-Sim is used to simulate NVIDIA GPU architectures and relies on the CUDA runtime environment. Within the LEGOSim framework, its *sendMessage*() and *receiveMessage*() functions are mapped to CUDA *cudaMemcpy*(), facilitating data transfer between this simlet and others. In terms of timing synchronization, GPGPU-Sim records local clock by *gpu_sim_cycle* and *gpu_tot_sim_cycle* and updates them according to the target clock cycles to be advanced.

Integration of SCALE-Sim [52]. SCALE-Sim is integrated into LEGOsim as simlet through executing the corresponding python script with designated chiplet identifiers, topology, the workload of NPU as input parameters. As SCALE-Sim has only timing model, the functional model is implemented in a dedicated C++ model. It receives input through *receiveMessage*() and transmits output via *sendMessage*() as wrappers. Upon completion of the simulation, the wrapper proceeds reading the execution time from SCALE-Sim's output logs. This execution time will be added to the time get from *readSync*() and sent to other chiplets through *writeSync*(). Data is received using *receiveSync*() and *read_data*() and sent using *sendSync*() and *write_data*().

By standardizing APIs, inter-chiplet communication data management, and clock synchronization, the UII enables seamless interoperability between diverse simlets, reducing integration complexity.

5 Evaluation

5.1 Experimental Setup

The experiments were performed on a 20 cores Intel(R) Xeon(R) Gold 6133 CPU with 2.50GHz and 512G main memory server. The benchmarks include parallel convolution (conv) [39], breadth-first search (BFS) [14], matrix multiplication (matmul) [12], MLP [64], ResNet [24] and Transformer [23]. Following architectures were configured in the experiments: CPU-4GPU-NPU-3CiM, CPU-20GPU-15NPU, CPU-3GPU, CPU-DSA-CiM-7GPU, CPU-DSA-CiM-47GPU, CPU-DSA-CiM-97GPU and CPU-20GPU-15NPU. Sniper [25], GPGPU-Sim [37], a custom-developed simulator mimicking the architecture of the Eyeriss NPU, SCALE-Sim, and MNSIM were used as simlets for the CPU, GPU, domain-specific accelerator (DSA), NPU, and compute-in-memory (CiM), respectively. These heterogeneous multi-chiplet systems cannot be simulated by most of the existing simulators listed in Table 1, except for LEGOsim. The detailed configurations of each simulator are provided in Table 2. Two memory protocols were configured in these experiments: HBM3 and DDR5, both with capacity of 24GB [1] [3]. The thermal parameters of the interposer, as well as the core area and pitch of chiplets, are listed in Table 3.

Table 4: Configurations of SIMBA and CIM-based Accelerator.

Multi-chiplet System Architecture			
SIMBA		**CiM-based Accelerator**	
Number of PEs	16	Activation Buffer	150KB
Technology	16 nm FinFET	CiM Array Size	144KB
Voltage	0.42–1.2 V	Clock Frequency	100MHz
PE Clock Frequency	0.16–2.0 GHz	CiM Type	ReRAM
Global PE Buffer Size	64 KiB	CiM Array Performance	1024 MACs per cycle
Routers Per Global PE	3	Die-to-die Connections	1.2Gbps/link
NoC Bandwidth	68 GB/s/PE		
Microcontroller	RISC-V		

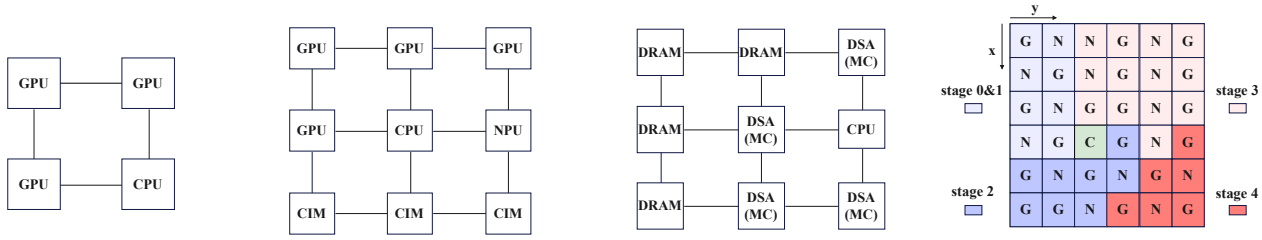
The transmission delay between adjacent chiplets is composed of the following three parts: 1) packetization and depacketization times (the values are obtained from [54] and [43]); 2) the transceiver latency (the values are obtained from [21] and [63]); and 3) the interposer wire delay and power models adopted from [31].

The inter-chiplet network topologies used in the experiment are mesh, meshLL (mesh with nodes (x,y) to node (x + 1, y + 1) connected by a long serial link) [20], NVL (a fat tree mimicking the NVlink structure), star, and torus.

5.2 Validating Simulation Accuracy and Analyzing Synchronization Overhead

To validate the fidelity of the simulator, the 4-chiplet, 8-chiplet, and 32-chiplet SIMBA [55] architectures, as well as the 4-chiplet, 5-chiplet, 9-chiplet, and 18-chiplet CiM-based accelerator [13], were simulated. In the CiM-based accelerator, each chiplet has CiM units using ReRAM, on-chip SRAM buffers, and high-speed interconnections. The chiplets' configurations in SIMBA and the CiM-based accelerator are detailed in Table 4.

The ResNet-50 benchmark runs on the 4-chiplet, 8-chiplet, and 32-chiplet SIMBA architecture, while the Tiny-Yolo [33] benchmark runs on the 4-chiplet, 5-chiplet, 9-chiplet, and 18-chiplet CiM-based accelerator to compare its performance with the reported data from these two references.



(a) CPU-3GPU (b) CPU-4GPU-NPU-3CIM (c) CPU-4DSA-4DRAM (d) CPU-20GPU-15NPU

Figure 8: Inter-chiplet network topologies of the multi-chiplet architectures in experiments.

1) To quantify the simulation error of SIMBA architecture, the error ε is defined as follows,

$$\varepsilon = \frac{|T_{sim} - T_{ref}|}{max\{T_{sim}, T_{ref}\}} \times 100\% \quad (1)$$

where T_{sim} and T_{ref} are simulated execution cycles and referenced execution cycles in [55] respectively.

The ε were 2.52%, 3.51% and 5.35% for 4-chiplet, 8-chiplet,and 32-chiplet systems respectively for the SIMBA simulation as illustrated in Table 5, which are quite low.

Table 5: Simulation accuracy validation.

SIMBA Multi-chiplet Architecture			
Architecture	4-chiplet	8-chiplet	32-chiplet
ε (%)	2.52	3.51	5.35
CiM-based Multi-chiplet Accelerator			
Architecture	4-chiplet	5-chiplet	9-chiplet	18-chiplet
ε_u (%)	2.71	4.68	2.69	5.79

2) To quantify the simulation error of the Tiny-Yolo model running on the CiM-based accelerator architecture [13], simulation error ε_u is defined as follows,

$$\varepsilon_u = \frac{|U_{sim} - U_{ref}|}{max\{U_{sim}, U_{ref}\}} \quad (2)$$

where U_{sim} and U_{ref} are simulated computing utilization and referenced [13] computing utilization receptively.

The ε_u were 2.71%, 4.68% , 2.69% and 5.79% for 4-chiplet, 5-chiplet, 9-chiplet and 18-chiplet systems respectively for the CIM-based accelerator as illustrated in Table 5, which are quite low. The low errors validate the high fidelity of LEGOSim in accurately modeling system performance.

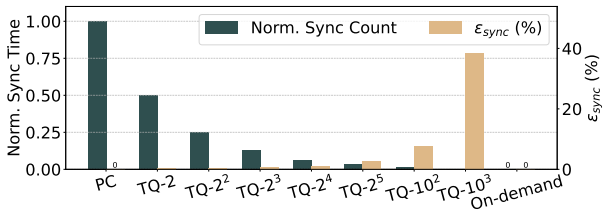


Figure 9: Comparison of synchronization event counts of PC, TQ, and OD synchronization methods.

5.3 Synchronization Time Comparison

Figure 9 compares the time of the proposed on-demand synchronization (OD) with per-cycle synchronization (PC) and time quantum synchronization (TQ) to simulate a-64 core system running the MLP benchmark. The synchronization time of the nine synchronization algorithms is normalized to that of PC. The inter-chiplet interconnection network topology is shown in the Figure 8a. TQ-x refers to synchronization occurring every x cycles. The OD approach reduces synchronization time by 99.9%, 99.9%, 99.8%, 99.7%, 99.4%, 98.1%, 96.6%, and 66.1% compared to PC, TQ-2, TQ-2^2, TQ-2^3, TQ-2^4, TQ-2^5, TQ-10^2, and TQ-10^3, respectively. Notably, TQ-10^3 exhibits a high synchronization error, whereas OD achieves high accuracy. The synchronization error quantifies the error with different synchronization methods w.r.t. PC, which is defined as:

$$\varepsilon_{sync} = \frac{|T_n - T_{pc}|}{max\{T_n, T_{pc}\}} \quad (3)$$

where $n \in \{TQ - x, OD\}, x \in \{2, 2^2, 2^3, 2^4, 2^5, 10^2, 10^3\}$. Here, T_n is total execution time with synchronization algorithm n. T_{pc} is the total execution time in PC synchronization. ε_{sync} are 0% for OD, and 0%, 0.04%, 0.12%, 0.66%, 0.87%, 1.9%, 8.6%, and 37.9% for PC, TQ-2, TQ-2^2, TQ-2^3, TQ-2^4, TQ-2^5, TQ-10^2, and TQ-10^3, respectively. These results indicate that as the synchronization interval in the TQ algorithm increases, ε_{sync} also increases. In contrast, on-demand synchronization exhibits the lowest overhead while maintaining high accuracy.

Figure 10 shows the time breakdown of sequential simulation, PC, and OD. The time of the three synchronization methods is normalized to the total simulation time of sequential simulation. Sequential simulation exhibits the highest chiplet-simulation time. PC

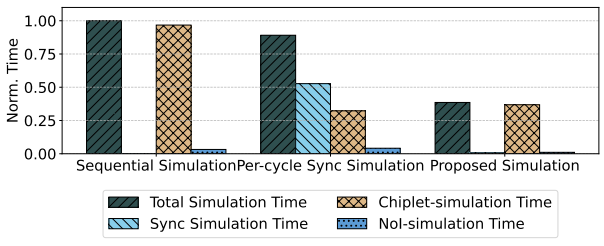


Figure 10: Simulation time comparison of three different methods.

reduces simulation time but incurs both the highest synchronization time. In contrast, OD has the lowest synchronization overhead and the lowest total simulation time.

Compared to the chiplet-simulation time in sequential simulation and the synchronization time in PC, the chiplet-simulation and synchronization time with OD are reduced by 61.9% and 98.1%, respectively. Furthermore, the total simulation time of LEGOSim is reduced by 61.4% and 56.7% compared to sequential simulation, and per-cycle synchronized parallel simulation, respectively.

5.4 Scalability Analysis

LEGOSim can be used to simulate large-scale multi-chiplet systems on a single server. Table 6 compares the simulation times of 10-chiplet, 50-chiplet, and 100-chiplet systems, all running the same input sized MLP benchmark. The simulation times are normalized to that of the 100-chiplet system. All three systems adopt a mesh inter-chiplet interconnection network topology. The 10-chiplet, 50-chiplet, and 100-chiplet configurations are CPU-DSA-CiM-7GPU, CPU-DSA-CiM-47GPU, and CPU-DSA-CiM-97GPU, respectively. The simulation times of the 50-chiplet and 10-chiplet systems are 46.6% and 12.3% of the 100-chiplet system's simulation time, respectively. In contrast, existing simulator cannot flexibly simulate heterogeneous multi-chiplet system up to 100 chiplets.

Table 6: Simulation time comparison

Architecture	10-chiplet	50-chiplet	100-chiplet
Norm. Time	0.12	0.47	1

6 Case Studies

6.1 Exploring the Design Space of On-chip Buffer and Inter-chiplet Interconnection Network

In the first case study, we conducted a design space exploration (DSE) using LEGOsim. The experiment was configured on a CPU-20GPU-15NPU architecture with a mesh topology as inter-chiplet interconnection network, as illustrated in Figure 8d, where "C", "G" and "N" are CPU, GPU and NPU chiplets, respectively. The ResNet-50 benchmark was the workload. In the baseline configuration, each GPU chiplet has 50 Streaming Multiprocessors. The NPU chiplet adopts the SIMBA architecture. Additional configuration details are provided in Table 7. The NoI bandwidth of this multi-chiplet architecture is 100 GB/s.

In this setup, the 36 chiplets are divided into four groups, with each group computing one or two stages of ResNet-50. To identify the performance bottlenecks of this architecture when running ResNet-50, running the ResNet-50 benchmark to this multi-chiplet system involves following three steps: allocating tasks to different chiplets, inserting the inter-chiplet communication (using the API functions defined in Section 4) and synchronization.

In the first step, tasks are assigned to different chiplets based on their computational workloads. Layer res2[a-c]_branch2c, res[2-5]a_branch1, res3[a-d]_branch2c, res4[a-f]_branch2c, and res5[a-c]_branch2c of ResNet-50 are allocated to NPU chiplets. Other layers are allocated to GPU chiplets. In Figure 8d, res1 through res5 correspond to stages 0 through 4, respectively. The CPU chiplet is the manager, distributing computation tasks to other chiplets.

Table 7: Configurations of the CPU-20GPU multi-chiplet systems

GPU chiplet		CPU chiplet	
# of SMs	50	# of Cores	8
Technology	4nm FinFET	Technology	7nm FinFET
L1 Cache Size	128KB	L1 Cache Size	512KB
Architecture	Nvidia Hopper	L2 Cache Size	4MB
L2 Cache Size	50MB	L3 Cache Size	16MB
Frequency	2GHz	Base Frequency	3.2GHz

In the second step, the tasks running on the GPU chiplets are programmed using CUDA. The tasks on the NPU chiplets are implemented by configuring a CSV topology file in SCALE-Sim. This topology file defines the layers of the workload. In SCALE-Sim, convolution layers and other operations that can be expressed in terms of equivalent GEMM operations are described using the M, N, K format in the workload topology. The tasks running on the CPU chiplets are programmed using C++.

Table 8: Performance comparison

	Computation	Buffer access	NoI
Norm. Time	0.34	0.72	1

As shown in Table 8, the Network-on-Interposer (NoI) latency and on-chip buffer access time are identified as the performance bottlenecks in this case. The times of computation, buffer access, and NoI are normalized to that of NoI latency. For example, chiplet (0,0) spent 35.6% and 42.9% time in buffer access and waiting for the remote data access. The breakdowns of a few chiplets' performances, which are normalized to the NoI latency of chiplet (3,0), are shown in Figure 11. In what follows, the on-chip buffer size and NoI bandwidth are selected as design variables to reduce the overall execution time.

To model the impact of on-chip buffer size and NoI bandwidth w.r.t. execution time, LEGOsim is run with different configurations. The following performance model is obtained using the maximum likelihood method [47]:

$$T = d + exp(a - b\ln(I+1) - c\ln(B+1)) \quad (4)$$

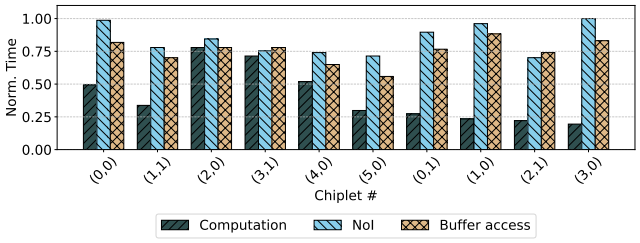


Figure 11: Breakdown of the performances for selected chiplets.

Table 9: Configurations of reference architectures

Power limit (W)	Reference configurations 1		Reference configurations 2	
	Buffer size (MB)	NoI bandwidth (GB/s)	Buffer size (MB)	NoI bandwidth (GB/s)
6200	2	512	2	512
6300	8	512	10	512
6400	15	1024	17	1024
6500	23	1024	24	1024
6600	30	1024	33	1024
6700	40	2048	45	2048

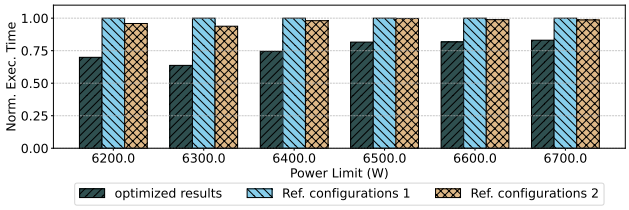


Figure 12: Execution time comparison by varying power budgets.

where a, b, c and d are regression coefficients and I, T, B are NoI bandwidth, total execution time, and buffer size of each chiplet. Eqn. 4 has a regression error of 8%.

To explore the design space, an optimization problem is defined to minimize the execution time under power constraint with power models adopted as in [62]. NSGA-II [19] is used to solve this problem. For comparison, two reference architectures listed in Table 9 are used. Figure 12 shows that, under different power budgets, the proposed solution achieves the lowest execution time. For example, it reduces execution time by 30% and 27% compared to reference configurations 1 and 2 under a power budget of 6109 W, respectively. Execution time of each configuration is normalized to that of the maximum execution time of reference configurations 1 and 2. This example shows that LEGOsim can be used to identify performance breakdowns and bottlenecks, generate datasets with different configurations for performance modeling, which is used in design space exploration (i.e., optimizing performance under power constraints).

6.2 Alleviating Computation Bottlenecks Using LEGOsim

In this case study, we demonstrate how LEGOSim can be used to flexibly and accurately compare various multi-chiplet architectures to identify and address computational performance bottlenecks and trade-offs inherent to these architectures. Initially, a baseline architecture CPU-4GPU-NPU-3CiM, connected via a 3×3 mesh inter-chiplet network, was configured. This setup, referred to as the CPU-4GPU-NPU-3CiM architecture, was tasked with running the parallel convolution benchmark with a convolution matrix of size $128 \times 128 \times 3$, are shown in Figure 8b.

To analyze performance, a key metric is defined, $\tau_{(x,y)}$ (computation-to-communication-latencyratioofchiplet(x, y)), as the ratio of each chiplet's execution time to its communication latency. Figure 13 reveals that $\tau_{(0,0)}$, the computation-to-communication-latency ratio of the GPU chiplet at (0,0), reaches the highest value of 11.5. Indicating that the GPU chiplet at (0,0) is the bottleneck in terms of computation.

To address this issue, we reconfigured the system by adding two additional GPU chiplets and redistributing the workload previously handled by GPU (0,0). After this adjustment, $\tau_{(0,0)}$ is reduced to 7, and the overall system execution time is decreased by 15%.

This case study highlights the effectiveness of LEGOSim for evaluating the performance of different multi-chiplet architectures.

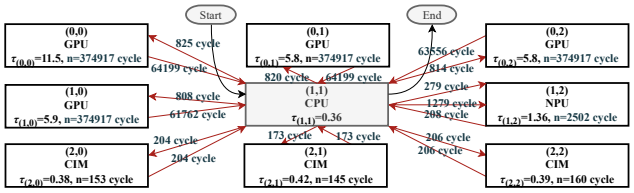


Figure 13: Chiplet level task graph of the parallel convolution benchmark with CPU-4GPU-NPU-CiM architecture, where n is execution time.

6.3 Evaluating Different Inter-chiplet Network Topology Configurations

For our first case study, LEGOSim was used to evaluate the impact of different inter-chiplet network topologies on the multi-chiplet system. Using the CPU-4GPU-DSA-CiM architecture, LEGOSim was configured with various inter-chiplet network topologies, including mesh, meshLL, NVL, and torus. These configurations were evaluated using benchmarks such as matmul, MLP, and Transformer, with varying packet flit sizes.

Figure 14 compares the normalized execution times with different inter-chiplet network configurations. With a flit size of 4, the matmul benchmark achieves the shortest execution time. The

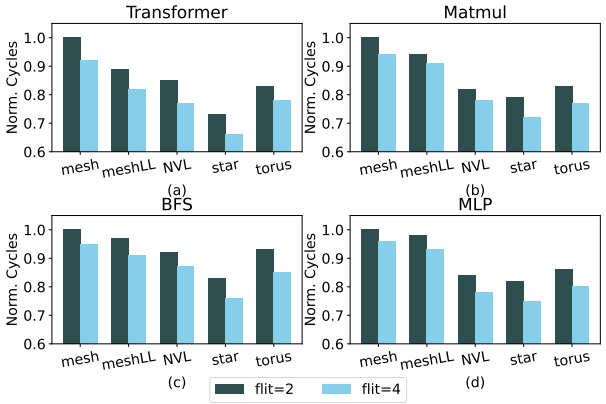
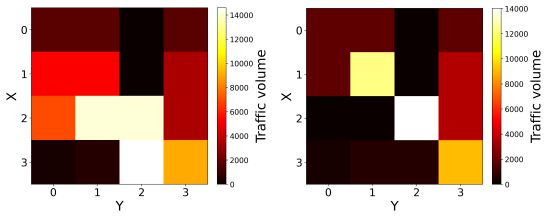


Figure 14: The execution times of (a) Transformer, (b) Matmul, (c) BFS, (d) MLP with different inter-chiplet network configurations.

execution times for the transformer, matmul, BFS, and MLP benchmarks were reduced by 7%, 5.2%, 6.2%, and 5.6%, respectively, when the flit size increases from 2 to 4.

A visualization tool for inter-chiplet traffic distribution of each D2D interface is included in LEGOSim as shown in Figure 15. Through this tool, researchers can observe the traffic volume at each D2D interface and the number of packets transmitted between chiplets, which can help researchers to find out the bottleneck of the multi-chiplet system more easily.



(a) mesh (b) meshLL

Figure 15: Inter-chiplet network traffic distribution of the matmul benchmark with (a) mesh and (b) meshLL as inter-chiplet network topologies.

6.4 Evaluating HBM3 vs. DDR5 in a CPU-4DSA-4DRAM Multi-chiplet System

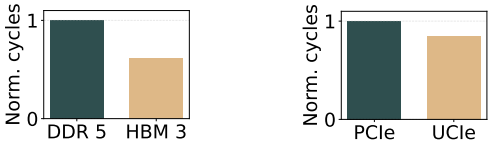
For this case study, we examine the impact of different memory protocols (HBM3 versus DDR5) in the CPU-4DSA-4DRAM multi-chiplet system, using ResNet-50 as benchmark, where a DDR DRAM with 32 GB is connected to the memory controller in the CPU chiplets. The inter-chiplet interconnection topology is mesh as shown in Figure 8c where each DSA has a memory controller (MC) and UCIe is used as D2D communication protocol.

Figure 16a shows that the total execution cycle of the system with HBM 3 is 39.1% lower than that of the system with DDR 5. The significant performance improvement demonstrates that HBM 3 is a superior choice for bandwidth-intensive workloads, particularly for deep learning inference tasks. These results, obtained through LEGOSim, reinforce its capability to accurately model memory hierarchy trade-offs in multi-chiplet architectures, making it an effective tool for guiding system design decisions.

6.5 Evaluating UCIe vs. PCIe in a CPU-4DSA-4DRAM Multi-chiplet System

Beyond memory protocols, D2D interconnection technology plays a pivotal role in determining overall system performance. This case study evaluates the impact of adopting Universal Chiplet Interconnection Express (UCIe) [56] and Peripheral Component Interconnection Express (PCIe) [5] as the D2D communication protocol in a 1CPU-4DSA-4DRAM multi-chiplet architecture. The inter-chiplet network topology is shown in the Figure 8c. LEGOSim was used to model and analyze both configurations to assess their impact on execution time, focusing on inter-chiplet interconnection protocol and communication time within the multi-chiplet system.

Figure 16b shows that the total execution time of the system with UCIe is 16.08% lower than that of the system with PCIe. These improvements highlight UCIe's ability to minimize interconnection latency, making it a more efficient solution for chiplet-based architectures.



(a) HBM 3 vs DDR 5 (b) UCIe vs PCIe

Figure 16: Performance comparision.

The findings further validate LEGOSim's ability to model interconnection trade-offs, demonstrating its effectiveness in evaluating chiplet design choices. By capturing the performance impact of different interconnection technologies, LEGOSim proves to be a valuable tool for optimizing next-generation multi-chiplet systems.

7 Conclusion

In this paper, we proposed LEGOSim, a modular and unified parallel simulation framework tailored for heterogeneous multi-chiplet systems. LEGOSim supports seamless integration of diverse simulators (simlets) as processes in parallel simulation, enabling accurate and flexible modeling. To address synchronization bottlenecks, on-demand synchronization was proposed, where synchronization occur only upon inter-chiplet communication to reduce synchronization overhead in parallel simulatoin. A decoupled simulation strategy was proposed to mitigate the inter-chiplet communication modeling overhead by decoupling chiplet simulation from inter-chiplet communication modeling. The Unified Integration Interface (UII) was proposed as a standard interface, allowing existing simulators like gem5, Sniper, and GPGPU-Sim to be integrated with minimal code changes to support parallel simulation. Experimental result shows that, LEGOSim has modeling errors of 3.79% and 3.94% when validating against SIMBA and a CiM-based accelerator, indicating high fidelity. LEGOSim also decreases synchronization overhead by 99.9% and 66.1% compared to per-cycle synchronization and time quantum, respectively. LEGOSim was showcased to analyze the performance bottleneck and perform design space exploration for various multi-chiplet systems. LEGOSim was open sourced, and hopefully can facilitate design space exploration for future large-scale multi-chiplet systems.

References

[1] [n. d.]. HBM3 IP Technical Bulletin. urlhttps://www.synopsys.com/designware-ip/technical-bulletin/hbm3-ip-dwtb.html.
[2] [n. d.]. Intel Distribution of OpenVINO toolkit. urlhttps://software.intel.com/en-us/openvino-toolkit.
[3] [n. d.]. SK Hynix Details Its DDR5-6400 DRAM Chip. urlhttps://www.anandtech.com/show/13999/sk-hynix-details-its-ddr56400-dram-chip.
[4] [n. d.]. The SR-71 of computing: Intel Ponte Vecchio retires after five years. urlhttps://www.jonpeddie.com/news/the-sr-71-of-computing-intel-ponte-vecchio-retires-after-five-years/.
[5] 2022. PCI Express Base Specification Revision 6.0 Version 1.0.

[6] 2025. LEGOSim. Link omitted to abide double-blind review policy; will be available in the final version.

[7] Niket Agarwal, Tushar Krishna, Li-Shiuan Peh, and Niraj K Jha. 2009. GARNET: a detailed on-chip network model inside a full-system simulator. In *Proc. IEEE Int'l Symp. Perform. Anal. Syst. Softw.* 33–42.

[8] AMD. 2023. *Zen 5 Architecture Overview.* Whitepaper.

[9] H. Angepat, D. Chiou, and E. S. Chung. 2014. *Simulator Background.* Springer International Publishing, 7–24.

[10] Yehia Arafa, Abdel-Hameed A Badawy, Gopinath Chennupati, Nandakishore Santhi, and Stephan Eidenbenz. 2019. PPT-GPU: scalable GPU performance modeling. *IEEE Comput. Archit. Lett.* 18, 1 (2019), 55–58.

[11] M. Badr and N. E. Jerger. 2014. SynFull: synthetic traffic models capturing cache coherent behaviour. *ACM SIGARCH Computer Architecture News* 42, 3 (2014), 109–120.

[12] Grey Ballard, Christopher Siefert, and Jonathan Hu. 2016. Reducing communication costs for sparse matrix multiplication within algebraic multigrid. *SIAM J. Sci. Comput.* 38, 3 (2016), 203–231.

[13] Jinshan Zhang Shunli Wang Xiaoyang Kang Lhua Zhang Mingyu Wang Bo Jiao, Haozhe Zhu and Chixiao Chen. 2021. Computing utilization enhancement for chiplet-based homogeneous processing-in-memory deep learning processors. In *Proc. Great Lakes Symp. VLSI.* 241–246.

[14] A. Buluç and K. Madduri. 2011. Parallel breadth-first search on distributed memory systems. In *Proc. SC Conf.* 1–12.

[15] G. Carneiro. 2010. NS-3: Network simulator 3. In *UTM Lab Meeting.* 4–5.

[16] Vincenzo Catania, Andrea Mineo, Salvatore Monteleone, Maurizio Palesi, and Davide Patti. 2015. Noxim: an open, extensible and cycle-accurate network on chip simulator. In *Proc. IEEE Int'l Conf. Appl.-Specific Syst., Archit. Processors.* 162–163.

[17] P. Y. Chen, X. Peng, and S. Yu. 2018. NeuroSim: a circuit-level macro model for benchmarking neuro-inspired architectures in online learning. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.* 37, 12 (2018), 3067–3080.

[18] J. Cubero-Cascante, N. Zurstraßen, and J. Nöller. 2023. parti-gem5: gem5's Timing Mode Parallelised. In *Proc. Int'l Conf. Embedded Comput. Syst.* 177–192.

[19] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. Evolutionary Computation* 6, 2 (2002), 182–197.

[20] Yinxiao Feng, Yuchen Wei, Dong Xiang, and Kaisheng Ma. 2024. Evaluating chiplet-based large-scale interconnection networks via cycle-accurate packet-parallel simulation. In *Proc. USENIX Annu. Tech. Conf.* 731–747.

[21] Yinxiao Feng, Dong Xiang, and Kaisheng Ma. 2023. Heterogeneous die-to-die interfaces: enabling more flexible chiplet interconnection systems. In *Proc. IEEE/ACM Int'l Symp. Microarch.* 930–943.

[22] J. J. Gómez-Hernández and E. F. Cassiraga. 1994. Theory and practice of sequential simulation. In *Geostatistical Simulations Workshop.* 111–124.

[23] Chaoyang He, Shen Li, Mahdi Soltanolkotabi, and Salman Avestimehr. 2021. PipeTransformer: Automated elastic pipelining for distributed training of transformers. *arXiv* (2021).

[24] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition.*

[25] W. Heirman, T. Carlson, and L. Eeckhout. 2012. Sniper: scalable and accurate parallel multi-core simulation. In *Proc. Int'l Summer School Adv. Comput. Archit.* 91–94.

[26] Intel. 2018. *OpenCL Beignet Project.* Technical Report. Intel.

[27] C. Jiang, A. Jayarajan, and H. Lu. 2023. Arbitor: a numerically accurate hardware emulation tool for DNN accelerators. In *Proc. USENIX Technical Conference.* 519–536.

[28] H. Jiang. 2022. Intel's Ponte Vecchio GPU: Architecture, Systems Software. In *Proc. IEEE Hot Chips Symp.* 1–29.

[29] Nan Jiang, George Michelogiannakis, Daniel Becker, Brian Towles, and William J Dally. 2010. *BookSim 2.0 user's guide.* Technical Report. Stanford Univ.

[30] M. Jung, J. Zhang, and A. Abulila. 2017. SimpleSSD: modeling solid state drives for holistic system simulation. *IEEE Computer Architecture Letters* 17, 1 (2017), 37–41.

[31] MD Arafat Kabir and Yarui Peng. 2020. Chiplet-package co-design for 2.5D systems using standard ASIC CAD tools. In *Proc. Asia South Pac. Des. Autom. Conf.* 351–356.

[32] Mahmoud Khairy, Zhesheng Shen, Tor M Aamodt, and Timothy G Rogers. 2020. Accel-sim: an extensible simulation framework for validated GPU modeling. In *Proc. ACM/IEEE Int'l Symp. Comput. Archit.* 473–486.

[33] Ivan Khokhlov, Egor Davydenko, and Ilya Osokin. 2020. Tiny-YOLO Object Detection Supplemented with Geometrical Data. *arXiv.*

[34] K. Shafie Khorassani, J. Hashmi, and C. H. Chu. 2021. Designing a ROCm-aware MPI library for AMD GPUs: early experiences. In *Proc. Int'l Conf. High Performance Computing.* 118–136.

[35] Hyesoon Kim, Jaekyu Lee, Nagesh B Lakshminarayana, Jaewoong Sim, Jieun Lim, and Tri Pho. 2012. *Macsim: a CPU-GPU heterogeneous simulation framework user guide.* Technical Report. Georgia Inst. Technol. 1–57 pages.

[36] Y. Kim, W. Yang, and O. Mutlu. 2015. Ramulator: a fast and extensible DRAM simulator. *IEEE Computer Architecture Letters* 15, 1 (2015), 45–49.

[37] Chao-Lin Lee, Min-Yih Hsu, Bing-Sung Lu, Ming-Yu Hung, and Jenq-Kuen Lee. 2020. Experiment and enabled flow for GPGPU-sim simulators with fixed-point instructions. *J. Syst. Archit.* 111 (2020), 101783.

[38] D. Lee, D. Hong, and W. Choi. 2022. MQSim-E: an enterprise SSD simulator. *IEEE Computer Architecture Letters* 21, 1 (2022), 13–16.

[39] Sunwoo Lee, Dipendra Jha, Ankit Agrawal, Alok Choudhary, and Wei-keng Liao. 2017. Parallel deep convolutional neural network training by exploiting the overlapping of computation and communication. In *Proc. IEEE Int'l Conf. High Perform. Comput.* 183–192.

[40] M. Leinhauser, J. Young, and S. Bastrakov. 2021. *Performance analysis of PIConGPU: particle-in-cell on GPUs using NVIDIA's NSight systems and NSight compute.* Technical Report. Oak Ridge National Laboratory.

[41] H. Li, J. Li, and A. Kaufmann. 2022. Simbricks: end-to-end network system evaluation with modular simulation. In *Proc. ACM SIGCOMM Conf.* 380–396.

[42] Shang Li, Zhiyuan Yang, Dhiraj Reddy, Ankur Srivastava, and Bruce Jacob. 2020. DRAMSim3: a cycle-accurate, thermal-capable DRAM simulator. *IEEE Comput. Archit. Lett.* 19, 2 (2020), 106–109.

[43] Xiaoyan Li, Zizheng Dong, and Shuaipeng Li. 2023. MUG5: Modeling of Universal Chiplet Interconnect Express (UCIe) Standard Based on gem5. *IEEE Int'l Conf. ASIC* (2023), 1–4.

[44] Q. Liu, M. Zapater, and D. Atienza. 2025. Gem5-acceSys: enabling system-level exploration of standard interconnects for novel accelerators. *arXiv* (2025).

[45] Jason Lowe-Power, Abdul Mutaal Ahmad, Ayaz Akram, Mohammad Alian, Rico Amslinger, Matteo Andreozzi, Adrià Armejach, Nils Asmussen, Brad Beckmann, Srikant Bharadwaj, et al. 2020. The gem5 simulator: version 20.0+. *arXiv Preprint* (2020).

[46] Jason E Miller, Harshad Kasture, George Kurian, Charles Gruenwald, Nathan Beckmann, Christopher Celio, Jonathan Eastep, and Anant Agarwal. 2010. Graphite: a distributed parallel simulator for multicores. In *Proc. Int'l Symp. High-Perform. Comput. Archit.* 1–12.

[47] In Jae Myung. 2003. Tutorial on maximum likelihood estimation. *J. Mathematical Psychology* 47, 1 (2003), 90–100.

[48] J. Power, J. Hestness, and M. S. Orr. 2014. gem5-gpu: a heterogeneous cpu-gpu simulator. *IEEE Computer Architecture Letters* 14, 1 (2014), 34–36.

[49] Y. M. Qureshi, W. A. Simon, and M. Zapater. 2019. Gem5-x: a gem5-based system level simulation framework to optimize many-core platforms. In *Proc. Simulation Conf.* 1–12.

[50] A. F. Rodrigues, K. S. Hemmert, and B. W. Barrett. 2011. The structural simulation toolkit. *ACM SIGMETRICS Performance Evaluation Review* 38, 4 (2011), 37–42.

[51] S. Rogers, J. Slycord, and M. Baharani. 2020. gem5-salam: a system architecture for LLVM-based accelerator modeling. In *Proc. Int'l Symp. Microarch.* 471–482.

[52] Ananda Samajdar, Yuhao Zhu, Paul Whatmough, Matthew Mattina, and Tushar Krishna. 2018. Scale-sim: systolic CNN accelerator simulator. *arXiv Preprint* (2018).

[53] D. Sanchez and C. Kozyrakis. 2013. ZSim: fast and accurate microarchitectural simulation of thousand-core systems. *ACM SIGARCH Comput. Archit. News* 41, 3 (2013), 475–486.

[54] Fabian Schätzle, Carlos Falquez, and Stefan Heinen. 2024. Modeling methodology for multi-die chip design based on gem5/SystemC co-simulation. In *Proc. Workshop on Rapid Simul. and Perform. Eval. for Design.* 35–41.

[55] Yakun Sophia Shao, Jason Clemons, Rangharajan Venkatesan, Brian Zimmer, Matthew Fojtik, Nan Jiang, Ben Keller, Alicia Klinefelter, Nathaniel Pinckney, Priyanka Raina, et al. 2019. Simba: Scaling Deep-Learning Inference with Multi-Chip-Module-Based Architecture. In *Proc. IEEE/ACM Int'l Symp. Microarchitecture.* 14–27.

[56] Debendra Das Sharma, Gerald Pasdast, Zhiguo Qian, and Kemal Aygun. 2022. Universal Chiplet Interconnect Express (UCIe): an open industry standard for innovations with chiplets at package level. *IEEE Trans. Compon. Packag. Manuf. Technol.* 12, 9 (2022), 1423–1431.

[57] Yifan Sun, Trinayan Baruah, Saiful A Mojumder, Shi Dong, Xiang Gong, Shane Treadway, Yuhui Bao, Spencer Hance, Carter McCardwell, Vincent Zhao, et al. 2019. MGPUSim: enabling multi-GPU performance modeling and optimization. In *Proc. Int'l Symp. Comput. Archit.* 197–209.

[58] Rafael Ubal, Byunghyun Jang, Perhaad Mistry, Dana Schaa, and David Kaeli. 2012. Multi2Sim: a simulation framework for CPU-GPU computing. In *Proc. Int'l Conf. Parallel Archit. Compil. Tech.* 335–344.

[59] A. Varga. 2010. *OMNeT++.* Springer Berlin Heidelberg, 35–59.

[60] Jun Wang, Jesse Beu, Rishiraj Bheda, Tom Conte, Zhenjiang Dong, Chad Kersey, Mitchelle Rasquinha, George Riley, William Song, He Xiao, and other. 2014. Manifold: a parallel simulation framework for multicore systems. In *Proc. IEEE Int'l Symp. Perform. Anal. Syst. Softw.* 106–115.

[61] Xiaohang Wang, Yifan Wang, Yingtao Jiang, Amit Kumar Singh, et al. 2025. On Task Mapping in Multi-chiplet Based Many-core Systems to Optimize Inter-and Intra-chiplet Communications. *IEEE Trans. Computers* 74, 2 (2025).

[62] X. Wang, M. Xu, A. K. Singh, Y. Jiang, and M. Yang. 2025. On Optimizing Inter- and Intra-Chiplet Interconnection Topologies for Robust Multi-Chiplet Systems.

IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems (2025).
[63] Bingyi Ye, Kai Sheng, and Weixin Gai. 2023. A 2.29-pJ/b 112-Gb/s Wireline Transceiver With RX Four-Tap FFE for Medium-Reach Applications in 28-nm CMOS. *IEEE J. Solid-State Circuits* 58, 1 (2023), 19–29.
[64] H. Zhang. 2018. Distributed deep learning training with Horovod. *arXiv* (2018).
[65] Zhenhua Zhu, Hanbo Sun, Tongxin Xie, Yu Zhu, Guohao Dai, Lixue Xia, Dimin Niu, Xiaoming Chen, Xiaobo Sharon Hu, Yu Cao, et al. 2023. MNSIM 2.0: a behavior-level modeling tool for processing-in-memory architectures. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.* 42, 11 (2023), 4112–4125.
[66] L. Zuolo, C. Zambelli, and R. Micheloni. 2017. Ssdexplorer: a virtual platform for SSD simulations. *Solid-State-Drives (SSDs) Modeling: Simulation Tools Strategies* (2017), 41–65.