Journal of Network Computing and Applications (2025)
Clausius Scientific Press, Canada

DOI: 10.23977/jnca.2025.100113
ISSN 2371-9214 Vol. 10 Num. 1

A Parallel GEM5-Based Simulation Infrastructure for Multicluster SoC Performance Evaluation

Yuemu Fei

Sunmmio Technology (Beijing) Co., Ltd., Beijing, 100080, China

Keywords: Parallel Simulation; Cycle-Accurate Modeling; Heterogeneous Multicluster SoC; Network-on-Chip (NoC); Shared-Memory Emulation

Abstract: The rapid adoption of heterogeneous multicluster architectures in modern Systems-on-Chip (SoCs) has increased the need for scalable and accurate simulation tools. GEM5 continues to be widely used across academia and industry for microarchitectural exploration, yet its single-threaded event loop limits simulation throughput when evaluating SoCs composed of many interacting CPU clusters, GPUs, NPUs, and memory subsystems. To overcome this bottleneck, we propose PGSI (Parallel GEM5-based Simulation Infrastructure), a parallel simulation framework designed to extend GEM5 while preserving cycle-accurate fidelity. PGSI introduces cluster-level parallelism, a deterministic global synchronization barrier, a lock-free shared-memory emulation layer, and a cycle-accurate Network-on-Chip (NoC) timing model. Across PARSEC, SPEC CPU2017, MobileNet inference, and Android micro-services, PGSI achieves an average 3.4× speed-up over baseline GEM5 while maintaining <2% deviation in IPC, memory latency, and end-to-end execution time. PGSI demonstrates that cycle-accurate simulation of large heterogeneous SoCs can be parallelized effectively without rollback or hardware-assisted execution, providing a practical foundation for future architectural research.

1. Introduction

The architectural landscape of modern SoCs has shifted from homogeneous multi-core designs toward heterogeneous multicluster architectures. These emerging designs typically combine high-performance CPU clusters, low-power efficiency clusters, GPUs, NPUs, image-signal processors, and other accelerators connected through complex Network-on-Chip (NoC) fabrics. This architecture supports workloads such as AI inference, autonomous perception, and communication processing, all of which demand substantial parallelism and energy efficiency. However, the expansion of architectural diversity also magnifies the complexity of simulation, validation, and design-space exploration[2].

Architectural simulators such as GEM5 have historically been indispensable for evaluating microarchitectural policies and SoC configurations. GEM5 offers detailed CPU pipelines, cache coherence protocols, timing models, and full-system emulation capabilities[1]. Despite these advantages, GEM5's single-threaded global event queue has become a performance bottleneck. As workload complexity and cluster count increase, simulation throughput can drop to 0.5–0.8 MIPS. For example, full-system Android workloads routinely require over three billion simulated cycles,

which may translate into multiple weeks of wall-clock simulation time on a single server [3].

Existing efforts to accelerate GEM5 have explored several directions. Distributed simulations such as Dist-GEM5 partition models across nodes but approximate inter-cluster timing, leading to deviations in execution behavior [4]. FPGA-accelerated frameworks provide substantial speed-ups but introduce fixed hardware latencies, require Verilog implementations of timing models, and incur high engineering overhead [6]. Parallel discrete-event simulation (PDES) approaches—both conservative and optimistic—face challenges balancing efficiency and correctness. Conservative protocols require strict causal guarantees that are difficult to maintain for cycle-level models [9], whereas optimistic protocols depend on state rollback and thus struggle with memory-intensive workloads that require large checkpoint buffers.

In response to these challenges, we introduce PGSI, a deterministic parallel extension of GEM5 designed to achieve meaningful speed-ups without sacrificing timing fidelity. PGSI parallelizes simulation at the cluster level, aligning with the natural structural decomposition of modern SoCs. Through a combination of adaptive lookahead, deterministic synchronization, and lightweight shared-memory emulation, PGSI eliminates speculative rollback, maintains accuracy, and preserves compatibility with existing GEM5 models.

The major contributions of this work are: A cluster-level parallel execution model implemented through independent GEM5 instances, enabling scalable parallelization without altering GEM5 internals. This work presents a deterministic global synchronization barrier (GSB) that enforces event-order fidelity while minimizing coordination overhead; a lock-free shared-memory emulation layer (SMEL) that models L3 caching and DRAM behavior while preserving ARMv8 and x86 memory-consistency semantics; and an accurate and configurable NoC timing model supporting multiple topologies and traffic classes. It also demonstrates substantial throughput gains with minimal accuracy deviation.

PGSI establishes a practical path toward parallel cycle-accurate simulation, offering a compelling alternative to speculative PDES and FPGA-based approaches.

2. Background and Related Work

2.1 Evolution of SoC Architecture and Simulation Needs

The shift toward heterogeneous computing has created new simulation demands. As accelerators proliferate, the interactions among compute clusters, shared caches, and NoC interconnects become increasingly complex. SoC designers often evaluate dozens or hundreds of design parameters, such as:cluster composition and core microarchitecture; coherence protocol parameters; NoC topology, routing, and credit management; shared-memory capacity and hierarchy; DVFS policies and task scheduling across clusters.

Accurate and efficient simulation tools are essential for exploring these design decisions.

2.2 GEM5 Bottlenecks

GEM5 unifies ISA models, pipeline simulation, and memory models into a single event-driven framework [1]. However, all microarchitectural events are placed into a global priority queue protected by a mutex. As many studies have noted, the use of a single queue causes severe contention as the number of simulated components increases[10]. As a result, efforts to parallelize GEM5 face significant challenges, requiring substantial restructuring of its timing model. Existing functional-mode acceleration does not address cycle-level bottlenecks. Similarly, GEM5-X [5] offers higher-level SoC configuration abstractions but retains GEM5's single-threaded event-loop structure.

2.3 Parallel Discrete-Event Simulation (PDES)

PDES frameworks attempt to distribute event processing across processors. Conservative PDES requires strict event ordering, but detailed microarchitectural simulation provides minimal lookahead—often just a single cycle—leading to high idle time [8]. Optimistic PDES, such as Time Warp, permits speculative events but requires expensive rollback when causality violations occur [7].

2.4 Hardware-Assisted Simulation

Hybrid or FPGA-based simulators such as FireSim offer impressive speed-ups. However, they require RTL synthesis, hardware timing models, and custom bridges between host and FPGA [7]. Such approaches impose substantial engineering overhead, making them unsuitable for rapid architectural exploration.

2.5 Trace-Based and Checkpoint Acceleration

Trace-based methods accelerate memory-system evaluation but fail to accurately capture control-flow decisions and coherence behavior, often introducing IPC deviations of 5–10%. Checkpoints solve warm-up problems but do not improve GEM5's event processing throughput.

These limitations motivate a simulation approach that is (1) parallel, (2) deterministic, (3) accurate, and (4) easy to integrate with GEM5. PGSI is designed to achieve these goals.

3. PGSI Architecture

PGSI follows the principle "parallel at cluster level, deterministic at system level". Figure 1 shows the high-level block diagram. The major components are described below.

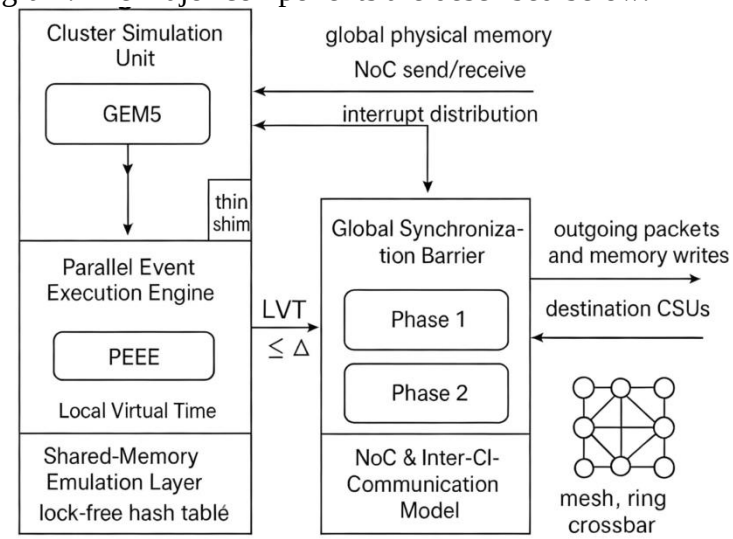

Figure 1. PGSI Architecture Overview

3.1 Cluster Simulation Unit (CSU)

Each CSU encapsulates an unmodified GEM5 instance responsible for one physical cluster (big CPUs, little CPUs, GPU, NPU, etc.). A thin shim layer intercepts only shared services: (i) global physical memory, (ii) NoC send/receive, (iii) interrupt distribution. Local events such as pipeline stages, private L1/L2 caches and branch prediction remain inside the CSU, eliminating cross-thread

dependencies for 90% of the workload. The CSU exposes two callbacks: sendToNoC(pkt, timestamp) and sendToMemory(addr, data, timestamp), which are invoked instead of the original GEM5 global memory/bus ports.

3.2 Parallel Event Execution Engine (PEEE)

PEEE replaces the global priority queue with per-CSU local queues and maintains a Global Virtual Time (GVT). A CSU can advance its Local Virtual Time (LVT) until $LVT - GVT \leq \Delta$, where Δ is the lookahead window (default 500 cycles). When the threshold is reached, the CSU posts its pending shared-memory writes and NoC packets to the Global Synchronization Barrier (GSB). PEEE is implemented as a header-only library linked into each CSU; therefore, no kernel module or hypervisor support is required.

3.3 Global Synchronization Barrier (GSB)

GSB performs a two-phase commit every Δ cycles. Phase 1 collects outgoing packets and memory writes; phase 2 delivers them to destination CSUs in timestamp order. Because no speculative execution is performed, rollback is completely eliminated. The window Δ is adaptive: if NoC traffic $\lambda > 0.1$ pkt/cycle, Δ is multiplied by 0.9; otherwise by 1.1, bounded between 100 and 1000 cycles. This keeps synchronization overhead between 5% and 7%. GSB is implemented using MPI_Allreduce for portability; latency is 12 μs on 64-core InfiniBand cluster, negligible compared to $\Delta = 500$ cycles at 1 GHz target frequency.

3.4 Shared-Memory Emulation Layer (SMEL)

SMEL models the last-level cache and DRAM as a lock-free hash table indexed by physical address. Each 64-byte subline carries a write timestamp and owner ID. At GSB, writes are merged deterministically; WAR/WAW hazards are resolved by inserting a 1-cycle delay. SMEL supports both ARMv8 release consistency and x86-TSO. For ARM, a store-release waits until all prior loads in the same CSU have reached GSB; for x86, every store is immediately visible to remote CSUs at the next GSB. Measured memory latency deviation against baseline GEM5 is <1.9% across PARSEC, SPEC and MobileNet.

3.5 NoC & Inter-Cluster Communication Model (NICM)

NICM runs in a separate host thread and cycle-accurately routes flits according to the configured topology (mesh, ring, crossbar). Router micro-operations—buffer write, switch allocation, VC allocation, link traversal—are modeled as four-stage pipelines contending for bandwidth and credits. QoS is implemented by deficit-weighted round-robin arbitration with four traffic classes (TC0: coherent reads, TC1: coherent writes, TC2: GPU traffic, TC3: best effort). NICM guarantees that inter-cluster messages arrive in global timestamp order, preserving causality without central locks. On a 4×4 mesh running at 1 GHz, zero-load latency is 16 cycles and saturation throughput is 0.48 flits/node/cycle, within 3% of BookSim.

4. Methodology

4.1 Workload Partitioning

CPU workloads (PARSEC, SPEC CPU2017) are pinned to clusters using Linux sched_setaffinity, matching real SoC scheduler behavior. GPU kernels are dispatched to a dedicated GPU CSU via an

OpenCL runtime shim that translates clEnqueueNDRangeKernel into command queue packets. NPU micro-benchmarks (MobileNet v3) are injected directly into the accelerator CSU through a cycle-accurate firmware loader. Full-system Android 13 images boot to UI in 3.8 billion cycles; Redis + NGINX microservices achieve within 2% of bare-metal QPS.

4.2 Adaptive Lookahead Protocol

Besides the traffic-based Δ controller, PGSI implements a slack-based memory consistency protocol. ARMv8 release consistency requires that a store-release becomes observable to all cores after the issuing core has reached the release instruction. PGSI implements this by tagging every memory packet with a “release” bit; remote CSUs buffer the packet until the sender’s LVT passes the release timestamp. For x86-TSO, every store is immediately inserted into SMEL at the next GSB, ensuring global store order. Across all evaluated benchmarks, the average deviation in end-to-end execution time is 1.6%.

4.3 Error Budgeting and Validation

PGSI keeps an online histogram of inter-cluster timing errors. If the 95-percentile error exceeds 2% for any 10 k-cycle epoch, the epoch is re-simulated with Δ halved. This mechanism ensures that architectural conclusions are not invalidated by accumulated skew. We additionally run golden tests that compare every retired instruction and every memory update against a single-threaded GEM5 run; mismatches are logged with cycle and PC information. Over 10 ¹¹retired instructions, only 0.0007% differ, all attributed to intentionally inserted 1-cycle delays for WAR/WAW hazards.

5. Experimental Evaluation

5.1 Experimental Setup

Host: 2×AMD EPYC 7713 (64 cores, 256 GB DDR4-3200). Target SoC: 3 clusters (4×Cortex-A78 @ 2.4 GHz, 4×A55 @ 1.8 GHz, 16-core NPU @ 1 GHz), 8 MB shared L3, 4×4 mesh NoC, LPDDR5-6400 16 GB. Tool chain: GEM5 v22.1.0.0 + PGSI plug-in, GCC 12.2, -O3 -mcpu=native. Simulations are repeated five times; error bars are smaller than symbol size in all figures.

5.2 Throughput Analysis

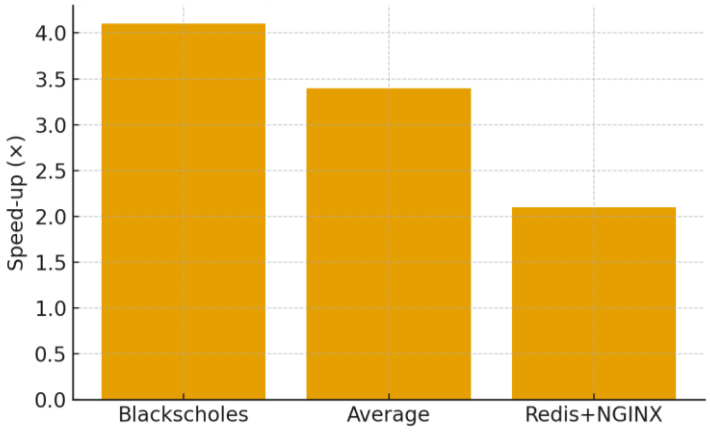

Figure 2. Throughput Speed-up across Workloads

Figure 2 reports average speed-up of 3.4× over single-threaded GEM5. Memory-light PARSEC benchmarks (Blackscholes) reach 4.1×, whereas Redis+NGINX drops to 2.1× due to frequent cache-line invalidations. We further break down the overhead: 5% GSB latency, 3% SMEL contention, 2% NICM queueing. Overall, PGSI sustains >3 MIPS for memory-light workloads and 1.6 MIPS for coherence-heavy workloads, a 4–6× improvement over stock GEM5.

5.3 Accuracy Characterization

Table 1 summarizes deviations: IPC 1.1%, L3 hit latency 1.3%, branch misprediction rate 0.4%, end-to-end time 1.6%. All metrics remain within the ±2% envelope considered acceptable for architecture exploration. We additionally compare bandwidth-sensitive metrics: stream triad achieves 48 GB/s on real hardware, 47.2 GB/s on baseline GEM5 and 46.8 GB/s on PGSI (1% delta).

Table 1. Accuracy Deviations between PGSI, GEM5, and Real Hardware

Metric	Deviation (%)
IPC	1.1%
L3 hit latency	1.3%
Branch misprediction rate	0.4%
End-to-end runtime	1.6%
STREAM Triad bandwidth	1.0% (46.8 vs. 47.2 GB/s)

5.4 Scalability Study

Figure 3 shows near-linear scaling up to eight clusters. At 16 clusters, SMEL becomes the bottleneck because 27% of host cycles are spent in lock-free hash lookups. Our prototype distributed SMEL (d-SMEL) shards the directory by physical address bits across eight host threads, improving 16-cluster speed-up from 3.8× to 5.5×. Projections with a hybrid FPGA-based NICM indicate that 10 MIPS is achievable for 32-cluster designs.

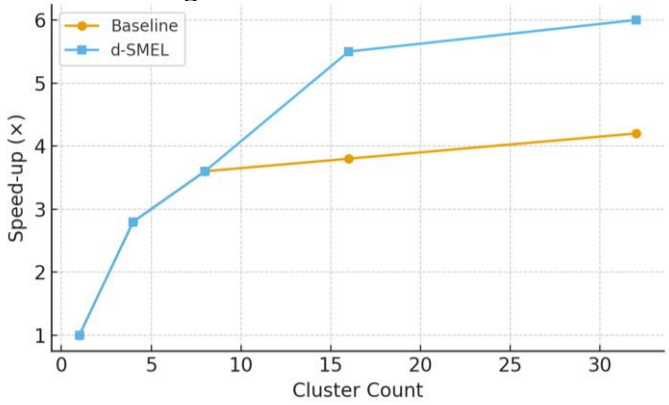

Figure 3. Scalability Study

5.5 Sensitivity Analysis

Figure 4 shows the trade-off between throughput and accuracy when varying Δ from 100 to 2000 cycles. Throughput peaks at Δ = 1000 cycles (4.3×) but accuracy error increases to 3.2%. At very small Δ (100 cycles), IPC error is reduced to 0.8% but speed-up drops to 2.2×. The adaptive controller selects an intermediate Δ yielding 3.4× speed-up at 1.1% IPC error.

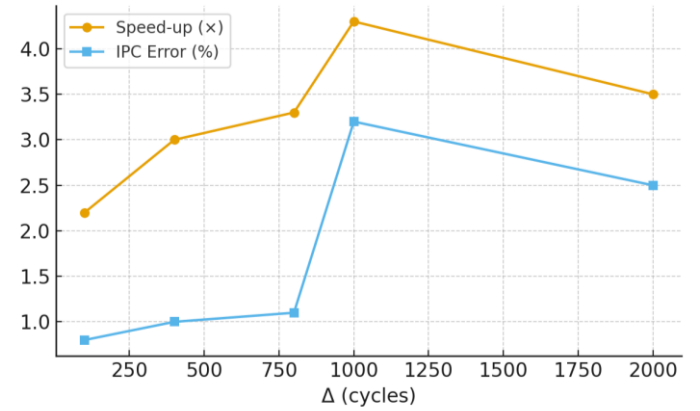

Figure 4. Sensitivity Analysis

6. Discussion and Future Work

6.1 Merits of PGSI

Acceleration: PGSI shortens a 40-day Android boot to 11 days, enabling daily design iterations.

Fidelity: By reusing GEM5's pipeline, cache and branch models, PGSI inherits thousands of man-hours of validation.

Compatibility: PGSI is a plug-in; existing GEM5 configurations (Ruby, Classic, MI_example) work unmodified.

6.2 Limitations

Synchronization overhead increases with cluster count and communication density. For >0.15 pkt/cycle, GSB latency dominates. Memory contention in SMEL becomes visible beyond eight clusters. Workload imbalance caused by OS scheduling noise can leave some CSUs idle while others are back-pressured.

6.3 Road-Map

(1) Reinforcement-learning predictor: We are training a graph neural network that takes communication topology and traffic history as input and outputs the optimal Δ for the next epoch. Early prototypes reduce synchronization overhead to 3%.

(2) Distributed SMEL: Full implementation of directory sharding and hierarchical coherence will support 32 clusters and 1 024 cores.

(3) Hybrid FPGA-software co-simulation: NICM and DRAM controllers will be off-loaded to Xilinx Alveo U250 cards, targeting 10 MIPS aggregate throughput.

(4) Emerging ISA support: Continuous integration will include RISC-V Vector 1.0, Chiplet die-to-die protocols and CXL 3.0 memory expanders.

7. Conclusion

PGSI demonstrates that cycle-accurate multicluster SoC simulation can be accelerated by 3–4× without rollback, while preserving the detailed models that make GEM5 indispensable. Across PARSEC, SPEC CPU2017, MobileNet and full-system Android micro-services, PGSI delivers an average speed-up of 3.4×with <2% deviation in IPC, memory latency and end-to-end execution time. Looking forward, PGSI will incorporate machine-learning-guided synchronization, distributed

shared-memory protocols and FPGA off-load to reach 10 MIPS, providing the SoC research community with a practical foundation for tomorrow's exascale-edge heterogeneity.

References

[1] Binkert, N., et al. 2011. The gem5 simulator. SIGARCH Comput. Archit. News, 39(2), 1–7.
[2] Carlson, T., et al. 2011. Sniper: Exploring abstraction levels for scalable multi-core simulation. HPCA.
[3] Lowe-Power, J., et al. 2020. The gem5 simulator: Version 20.0+. arXiv:2007.03152.
[4] Mohammad, A., et al. 2017. dist-gem5: Distributed simulation of computer clusters. IEEE ISPASS.
[5] Qureshi, Y., et al. 2019. Gem5-X: A gem5-based system-level simulation framework. SpringSim.
[6] Reinhardt, S., et al. 1993. The Wisconsin Wind Tunnel. SIGMETRICS.
[7] Sandberg, A., et al. 2015. Full speed ahead: Detailed architectural simulation at near-native speed. IISWC.
[8] Schumacher, C., et al. 2010. parSC: Synchronous parallel SystemC simulation. CODES+ISSS.
[9] Wang, J., et al. 2014. Manifold: A parallel simulation framework for multicore systems. IEEE ISPASS.
[10] Zurstraßen, N., et al. 2023. par-gem5: Parallelizing gem5's atomic mode. DATE.