# c2c-gem5: Full System Simulation of Cache-Coherent Chip-to-Chip Interconnects

Luis Bertran Alvarez[†‡] Ghassan Chehaibar[‡] Stephen Busch[‡] Pascal Benoit[†] David Novo[†]

[†]LIRMM, Univ. Montpellier, CNRS, Montpellier, France [‡]ATOS, Paris, France

***Abstract*—High-Performance Computing (HPC) is shifting toward chiplet-based System-on-Chip (SoC) architectures, necessitating advanced simulation tools for design and optimization. In this work, we extend the gem5 simulator to support cache-coherent multi-chip systems by introducing a new chip-to-chip interconnect model within the Ruby framework. Our implementation is adaptable to various coherence protocols, such as Arm CHI. Calibrated with real hardware, our model is evaluated using PARSEC workloads, demonstrating its accuracy in simulating coherent chip-to-chip interactions and its effectiveness in capturing key performance metrics early in the design flow.**

***Index Terms*—computer architecture simulation, HPC, cache coherency, chiplet, chip-to-chip interconnect.**

## I. Introduction

Driven by the need for cost-efficiency and improved yields, High-Performance Computing (HPC) is transitioning from traditional monolithic chip designs to chiplet-based System-on-Chip (SoC) architectures [1]. Accordingly, adequate simulation technology is required to tackle the new challenges in designing, evaluating, and optimizing these systems.

A particularly interesting chiplet-based design approach includes multiple cache-coherent multi-core chips that, when connected, maintain coherency across all interconnected chips. CCIX [2], UCIe [3] and OpenCAPI [4] are examples of open standards proposed to enable such systems. In this approach, a single-chiplet CPU serves edge applications, while multiple-chiplet CPUs deliver many-core cloud performance. Unfortunately, existing architecture simulators fall short in modeling such systems. They lack flexibility, detailed modeling, and OS control at the node level, limiting accurate simulation of modern coherent interconnect protocols [5].

Our goal in this work is to extend the gem5 simulator [6], a widely-used and open-source architecture simulator, to model cache-coherent multi-chip multi-processor systems. To this end, we design a new cache-coherent chip-to-chip (C2C) interconnect model within Ruby [7], a highly configurable cache coherence protocol simulator used within the gem5 simulation framework. Our interconnect model is designed to connect multiple chips that implement the Arm Coherent Hub Interface (CHI) coherence protocol [8], a state-of-the-art solution for high-performance, scalable, multi-core systems. Given the uncertainty about which chip-to-chip interconnect standard will become dominant, our goal is not to implement a specific protocol (e.g., CCIX). Instead, we aim to provide a reference implementation that can be adapted to any protocol.

We use a real hardware platform made up of two Arm N1SDP boards [9] connected via CCIX to calibrate our simulation model. We design a microbenchmark to measure the latency of local and remote main memory accesses. We evaluate our model running PARSEC [10] workloads on two CHI chips operating under the same OS, with a distributed, non-interleaved shared memory system. Our results include the observation of key metrics in a C2C scenario like the distribution of traffic type and bandwidth at the C2C link. We also show the execution overhead of using a coherent C2C link with real-life applications. Using hardware, we calibrate the model and show closely matching results, showcasing the flexibility of our simulator.

We make the following contributions in this paper:

- We identify the key challenges to extending CHI cache coherency to multiple chips.
- We introduce *c2c-gem5*, a new cache-coherent chip-to-chip interconnect model in gem5, and make our model freely available as open-source [11].
- We calibrate c2c-gem5 using a real hardware platform.
- We show results of c2c-gem5 running PARSEC workloads.

## II. Background

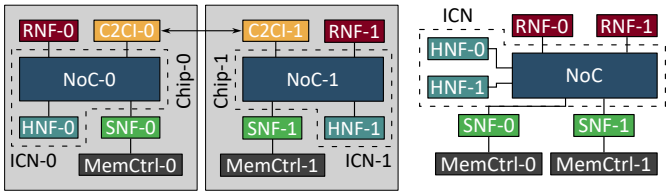
In multi-core systems, cache coherency is maintained by protocols that ensure all cores have a consistent view of memory. Arm CHI is a modern cache coherency protocol used in Arm [12] and RISC-V [13], [14] HPC architectures that support large many-core systems using a directory-based coherence mechanism. CHI supports MESI and MOESI cache coherence models and comprises three main components:

- The request node (RN) initiates transactions and sends memory requests. A fully coherent request node (RNF) caches data locally and must respond to snoop requests.
- The interconnect (ICN) serves as the responder for request nodes and encapsulates fully coherent home nodes (HNF), which act as points of coherency (PoC) and serialization (PoS) for specific address ranges. HNFs manage snoop requests to RNFs and memory access requests to SNFs.
- The subordinate nodes (SNF) interface with the memory controllers.

Figure 1 shows a simplified diagram of two chips, each including the necessary components to support local CHI coherency.

The CHI protocol includes four message types: (1) *request messages* initiate transactions such as read and write operations, (2) *snoop messages* maintain cache coherence by querying other caches for the state of specific data, (3) *response messages* provide the status or data needed to complete a transaction in response to requests and snoops, and (4) *data messages* carry the actual data between components.

978-3-9826741-0-0/DATE25/© 2025 EDAA




(a) (b)

Fig. 1: High-level C2C architecture (left) and reference architecture (right) with CHI components and C2CIs.

## III. Coherent Chip-to-Chip Interconnect

To enable global coherence across multiple CHI chips, we propose a new Chip-to-Chip Interface (C2CI) component along with minimal modifications to the HNF component, and additional metadata in the messages.

### A. On-chip vs. chip-to-chip requests

To demonstrate the key differences between a standard on-chip local memory request and one that requires traversing the chip-to-chip interface to access memory on another chip, we refer to Figure 1 and the following examples.

When Chip-0 is not connected to Chip-1, it operates following the standard CHI protocol. Thus, when RNF-0 initiates a local memory request, NoC-0 routes the request to HNF-0, which is responsible for managing memory coherency within Chip-0. HNF-0 monitors all caches that store a copy of the requested memory block (i.e., sharers). If the data is not found in any cache, HNF-0 requests SNF-0 to retrieve the data from Chip-0's main memory and to send it to RNF-0. Instead, if the data is present in a local cache, HNF-0 issues a snoop request (e.g., `SnpSharedFwd`) to that cache, which provides RNF-0 with the most up-to-date version of the data. Finally, HNF-0 updates the state of the block after receiving the corresponding acknowledgment messages.

When Chip-0 is connected to Chip-1 and RNF-0 initiates a memory request to a block allocated to Chip-1's main memory, new challenges arise. First, NoC-0 should check the address range and route all remote memory accesses to the chip-to-chip interface C2CI-0, instead of the local HNF. C2CI-0 then forwards the request to C2CI-1, as it belongs to the chip housing the HNF managing the address of the request. C2CI-1 initiates a new request that NoC-1 routes to HNF-1. In this case, HNF-1 must monitor all caches that may hold a copy of the requested block, both locally on Chip-1 but also remotely on Chip-0. When sharers are remote, HNF-1 must issue snoops that need to be forwarded by the C2CI-1 to Chip-0 and vice versa for the corresponding acknowledgments.

### B. Main design challenges

To extend CHI coherency across multiple chips, four main challenges must be addressed.

**Routing across chips**. The CHI NoC only routes messages to local components. Thus, for remote accesses, two C2CIs must coordinate to function as a gateway. In the previous example, C2CI-0 serves as the destination of the request in the local NoC-0, while C2CI-1 is the initiator of the forwarded request in the remote NoC-1. To achieve this, NoC routing tables must be updated to direct remote accesses to the local C2CI, and C2CIs need to incorporate both responder and initiator functionality.

**Tracking remote sharers**. The CHI HNF must track all caches storing a copy of a specific memory block to maintain data consistency across the system. When a cache modifies a shared memory block, the HNF identifies all caches holding copies of that block and issues snoop requests to either invalidate or update their copies. To extend CHI coherency across multiple chips, the HNF needs to be extended to track sharers located off-chip.

**Interactions between C2CIs and local chips**. For certain transactions, the behavior of the C2CI must differ based on whether the initiator is local or remote. For example, when a `SnpSharedFwd` request reaches the C2CI, it should expect a Snoop Response (`SnpResp`) if the request was initiated by a remote RNF, but it should expect an additional Data message (`Data`) if the request was initiated by a local RNF. To enable this differentiation, new metadata must be added to the CHI messages to differentiate between local and remote initiators.

**Chip-to-chip link management**. The C2CI must keep track of multiple in-flight transactions that are currently being processed by storing the necessary metadata (e.g., source, destination, type of transaction, etc.) locally in Transaction Buffer Entries (TBEs). This is straightforward when transactions operate on different addresses. However, it becomes significantly more complex when transactions operate on the same address. In such cases, the address is no longer sufficient to identify individual transactions, and more complex mechanisms are needed. One alternative is to limit the number of C2C in-flight transactions to the same address.

## IV. gem5 Implementation Challenges

We implement a new cache-coherent chip-to-chip interconnect model called *c2c-gem5* within gem5 [6], [15], a highly modular, cycle-accurate computer architecture simulator widely used in industry and academia. The gem5 simulator provides detailed models for CPUs, caches, memory and full-system simulation. It includes two different cache systems: Ruby, which models cache coherence protocols with high fidelity, and the "classic" caches, which lack cache coherence fidelity and flexibility. We build c2c-gem5 within Ruby using a domain-specific language called SLICC, which allows users to define the specific architecture and behavior of each controller and message type within a coherence protocol. Conveniently, Ruby includes an implementation of the CHI protocol developed by Arm [16].

However, implementing c2c-gem5 within Ruby is not straightforward, as Ruby does not natively support multiple interconnected NoCs. Ruby was originally designed to model monolithic chips including a single NoC model. But modeling multiple chips in Ruby requires multiple NoCs (see NoC-0

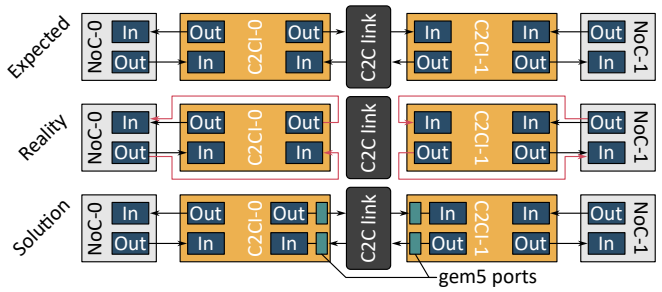



Fig. 2: Port connection problem when using Ruby ports between two controllers of different networks.

and NoC-1 in Figure 1). Recent work [17], [18] removed the single NoC limitation, enabling the use of multiple NoCs to model separate processor and GPU networks in the same Ruby environment. However, connecting multiple NoCs and extending their coherence is still problematic: Ruby makes implicit assumptions about how its controllers are connected to the NoC.

Figure 2 illustrates the problem. The top subfigure shows the expected configuration when connecting two Ruby controllers of different networks (e.g., C2CI-0 from NoC-0 and C2CI-1 from NoC-1). However, as shown in the middle subfigure, Ruby automatically connects all the ports of a controller to its corresponding network, including the C2C-side ports that we intend to connect to the C2C link. To address this issue, we use gem5 ports instead of Ruby ports to connect the C2CI to the C2C link, as shown in the bottom subfigure. In gem5, controllers outside of the Ruby environment use gem5 ports (i.e., classic ports) and gem5 packets for communication. For example, MemCtrl-0 in Figure 1 communicates with SNF-0 using a classic gem5 port. These bi-directional gem5 ports are fully controlled by the designer, avoiding the connectivity issues encountered with Ruby ports. Models outside of Ruby are addressed through gem5 classic ports, only one of such port, called `mem_out_port`, is available to Ruby controllers. This port, defined in the `AbstractController` class, is inherited by all Ruby controllers. Consequently, in order to declare a new gem5 classic port, we need to extend the AbstractController class, as well as the SLICC compiler.

## V. C2CI Implementation

Figure 3 shows a high-level overview of c2c-gem5. It features a new Ruby controller named C2C-Interface (C2CI) ①, and minimal modifications to the HNF component ② and the CHI message format ③. These elements work together to enable inter-chip communication within a single Ruby system and extend cache coherency across the chips.

**C2C Interface Controller**. The C2CI has multiple key responsibilities. First, it acts as an HNF for all local requesters. In Figure 1 the C2CI-0 is seen as HNF-1 by RNF-0. C2CI is a proxy for all off-chip address ranges. Second, it acts as a requester, injecting CHI request messages in the network in order to service requests initiated off-chip. Third, it sends chip-to-chip requests via the C2C link. Fourth, it receives chip-to-chip requests directed to the address space allocated to its on-chip HNF.

The C2CI is implemented as a SLICC controller, described using Finite State Machines (FSMs). Messages are sent by *actions* that are executed during *transitions* from one *state* to another. These transitions are triggered by *events*. The C2CI must support 73 different CHI transactions that can transit the C2C link. In order to keep the code to a reasonable length (approximately 2700 lines) we use two main mechanisms. First, we use generalized actions to handle different types of requests. For example, *sendToNet_Data* sends a data message to the network, independently of the type of data (e.g., `CompData_UC`, `SnpRespData_SC`, etc.). The fields of each message are filled with information from the TBE. Second, we sort the requests into different FSMs. Some requests are treated the same way, and can be executed using the same FSM. For example, *ReadShared* is treated the same way as *ReadUnique*.

The C2CI handles two different types of Ruby messages: the classic CHI message and a new C2C message (C2cMsg), as shown in Figure 3. We define C2cMsg using SLICC to carry requests, snoops, data, and responses over the C2C link. C2cMsg contains metadata serving two primary purposes: first, to facilitate transmission through the C2C link, and second, to enable the generation of appropriate CHI messages for injection into the destination NoC. This also includes CHI-specific fields like *RetToSrc* that affects whether or not a snoop response returns data, or *AllowRetry* if the request can be retried or not. Additionally, C2cMsg includes C2C-specific fields like *C2c_sharers* that indicate the off-chip sharer associated with an address for snoop routing. A *Priority* field is also included for C2C link access control to indicate that the CHI message is a priority snoop and should be treated, putting the ongoing transaction on hold. This link access control is used to manage the number of in-flight transactions, as indicated in Section III-B. Specifically, we implement a handshake mechanism that enables two controllers to acknowledge their readiness to process a request for a specific address.

Figure 4 shows a flow diagram of a message exchange for request acknowledging between two C2C-Interface controllers. In this example, the C2CI-0 is handling the address range containing the address of the request. This means that it has priority and does not need acknowledgment. In ❶ C2CI-0 receives a CHI request. Then, the controller determines if it is granted priority or not. This happens in the *Ack* block of Figure 3. In this example, blue request has priority so it is sent to the C2C link. In ❷ two things happen. First, C2C-1 checks if the yellow request has priority. It is not the case so C2CI-1 goes through the handshake process. The yellow request is sent to the C2C link with a special flag. Then, the initial request is enqueued in an internal queue to wait for the acknowledgment. This queue is labeled *stalled* in Figure 3. Second, the blue request is received by C2CI-1. The controller determines how the request is treated. In this example, the request has priority and is sent to NoC-1. In ❸

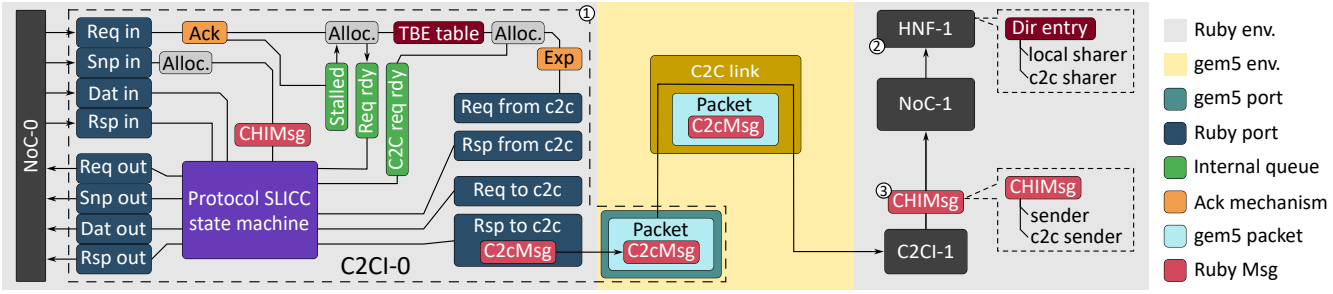



Fig. 3: High-level overview of c2c-gem5, including the C2CI controller internal structure.

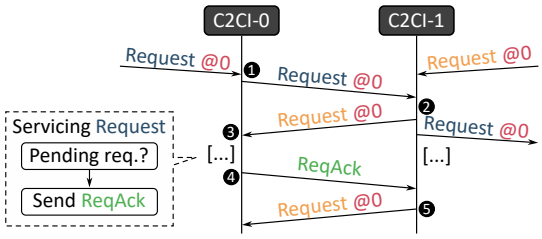



Fig. 4: C2C request handshake mechanism flow diagram.

C2CI-0 receives the yellow request. The request has the acknowledgment flag. Since C2CI-0 is treating the blue request the yellow request is not treated directly. A TBE flag for the pending request is set. In ❹, the blue request is finalized. C2CI-0 checks if there are pending requests and sends an acknowledgment message accordingly. In ❺, C2CI-1 receives the acknowledgment, dequeues the pending request of the stalled queue and processes it.

Finally, the C2CI controller has a different port structure for the network side and the C2C side. In Figure 3, network-side ports (on the left of the C2CI-0) are labeled as Ruby ports. These ports allow the C2CI to receive and send CHI messages to and from the local controllers (i.e., RNFs, HNFs). The C2C-side ports are also Ruby ports but are connected to gem5 ports. As indicated in Section IV, this architecture allows the C2CI to have available ports, which are not connected to the NoC, to receive and send C2C messages via the C2C link.

**Tracking off-chip sharers and initiators**. In addition to the C2CI, we introduce minor modifications to the HNF and CHI message format of the default gem5 CHI protocol implementation. The HNF sends various messages to RNFs to maintain cache coherence. Since the CHI implementation of gem5 is intended to work on monolithic chips, it lacks functionality to track the status of a cache line shared with off-chip RNFs. As such, the HNF will consider the C2CI, the controller initiating requests in the local NoC, as the sharer. This means that the HNF would only know that off-chip sharers have a copy of the cache line address, but it would be unable to identify the individual off-chip sharers. Consequently, the HNF would be unable to issue the necessary snoop requests to maintain coherence with off-chip sharers. To address this issue, we simply add a new metadata field called *c2c_sharers* in the directory data structure managed by the HNF. It allows the HNF to store the off-chip sharers for the cache line. Additionally, we add an off-chip sharer field to the metadata of the CHI and C2C messages to carry this information all the way to the HNF.

When the RNF receives the expected messages to complete a transaction, it sends an acknowledgment to the responder. Upon receiving this acknowledgment message, the C2CI-0 in Figure 3 forwards the message to the C2CI-1, populating a field of the message with the corresponding remote sharer of the line. The right side of the figure shows how the C2CI-1 will use this field to inject the CHI message in NoC-1 with the right off-chip sharer field. When the HNF-1 receives the acknowledgment and updates the directory entry, the field is used to also update the c2c_sharer field. Later, when the HNF-1 sends a snoop to the C2CI-1, it will populate a new field of the CHI message with the off-chip sharer, which will be used to route the snoop request to the appropriate chip.

Finally, we add a second field to the CHI message metadata to track remote initiators. This information is required by the FSM of the C2CI to properly handle certain incoming requests.

## VI. Experimental Setup

**Modeled architectures**. We execute gem5 22.1 on a server with an Intel Xeon Silver 4214R processor (2.4GHz) running CentOS Linux 7. We configure c2c-gem5 to model a *two-chip* architecture, as illustrated in Figure 1a. Each chip features a CPU with a 64 KB 4-way set-associative L1 data cache and a 1 MB 8-way set-associative L2 cache, one HNF, one SNF, and one C2C interface. Each chip also includes a memory controller connected to the SNF. Memory ranges are non-interleaved as follows: $[0, 1.5\ \text{GB})$ for Chip-0 and $[1.5\ \text{GB}, 3\ \text{GB})$ for Chip-1. Each memory controller includes one channel with 2 ranks, each rank containing 8 banks of DDR3-3200 MT/s. We also model a single-chip *reference* architecture that has the same characteristics but without the C2C link (Figure 1b). All controllers are assembled around a unique NoC. Both architectures are simulated in full-system mode, running Ubuntu Linux 18.04. The OS boots using *non-caching simple CPU* model to ensure fully deterministic simulation while reducing boot time by bypassing the cache subsystem. Upon entering the Region-of-Interest (ROI), we switch to the *timing CPU* model, which is more precise and compatible with the Ruby subsystem.

**Real hardware platform**. We use a real hardware testbed to calibrate the C2C model. It is built around two Quad-core N1 (Arm v8.2A) Software Development Platforms (N1SDP [9]). These two platforms use the Arm AMBA CMN-600 NoC protocol and are interconnected through a specialized PCIe riser card using the CCIX protocol.

**Workloads**. We evaluate c2c-gem5 using programs from the PARSEC [10] benchmark, selecting those that complete the simulation within 15 hours in the reference architecture. PARSEC primarily represents multi-threaded applications that demand large amounts of shared memory.

## VII. Results

**OS integration**. We validate the programmer's view of our multi-chip model. Using full-system simulation, we can boot the OS and run commands like the `lscpu` Linux command to gain insight into the underlying hardware. We execute this command on the simulated platforms shown in Figure 1, and the OS detects a 2-core processor in both the reference and the C2C architecture. When running the same command on the real hardware platform, we observe a consistent result, with the OS detecting an 8-core processor, while both interconnected platforms each consist of a 4-core processor.

**Microbenchmarking**. We use the real hardware platform described in Section VI to calibrate the latency of the chip-to-chip link in our model. We design a microbenchmark to measure DRAM access time using the following steps: (1) allocate a large array of 24 GiB (larger than the local DRAM capacity to force memory allocation into the remote memory), (2) randomly select a position in the array, (3) read data from the selected position, (4) flush the read data, (5) go to step 3 and repeat multiple times to reduce measurement noise, (6) measure the accumulated execution time and divide by the number of reads to obtain DRAM access time, (7) go to step 2 and repeat for a new address. Running this microbenchmark on the real hardware platform allows us to generate a histogram distribution of DRAM access times. The top graph in Figure 5 shows the resulting distribution of DRAM access times. Two distinct populations are identifiable: one centered around 110 ns and the other around 810 ns. The first population corresponds to local DRAM accesses, while the second represents off-chip DRAM accesses. We

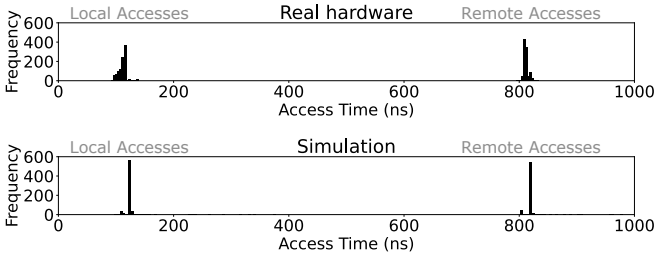



Fig. 5: Time distribution of local and remote accesses on the hardware testbed (top) and the calibrated simulation (bottom).

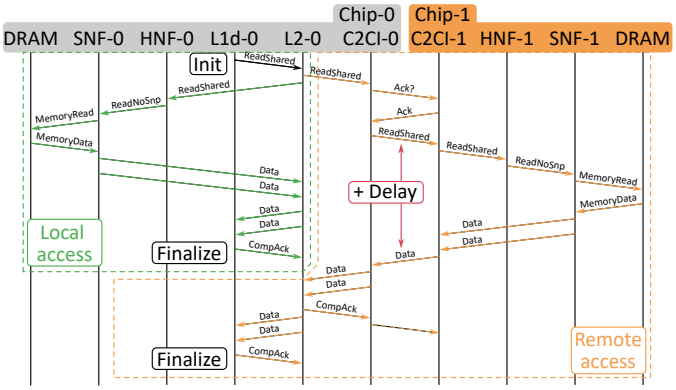


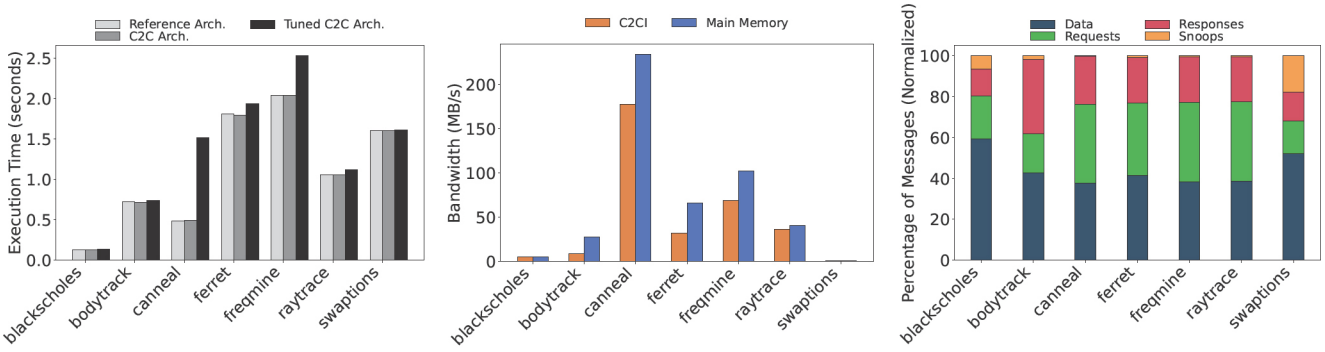

Fig. 6: Access flow diagrams for local (green) and remote (yellow) accesses.

validated this assumption using the Linux `pagemap` interface to inspect the corresponding physical addresses. We observe an 8× difference between local access (approximately 100 ns) and remote off-chip access (approximately 800 ns).

We use trace injection to reproduce the same experiment with the simulated C2C architecture. Using the gem5 traffic generator component, we can inject custom traffic into the memory subsystem. It allows us to target physical addresses providing full control over the access pattern. Figure 6 shows two different flow diagrams illustrating the transactions exchanged between Ruby controllers for local and remote DRAM accesses. On the left side, local addresses do not go through the C2C link. On the right side, we see the request being routed to Chip-1. The data is then routed back. The figure also highlights in red the delay added to calibrate the chip-to-chip link, emulating the behavior of the real hardware platform. The bottom graph of Figure 5 shows the distribution of DRAM access times in the simulated model after calibration. By comparing the two graphs, we conclude that the simulation model can easily be calibrated to precisely reproduce real hardware access times.

**Simulating multithreaded applications**. Figure 7 shows a range of results obtained from running PARSEC workloads on both the reference and chip-to-chip gem5 architectures described in Section VI.

Figure 7a shows the execution times for the reference and the C2C architectures before and after the chip-to-chip link calibration described in the microbenchmarking subsection. We make two key observations. First, the reference and baseline C2C architectures achieve very similar execution times. The minor differences are due to variations in request ordering caused by slight runtime variations. Second, the calibrated C2C architecture shows a noticeable slowdown in some programs, while others remain unaffected. In particular, `canneal` shows the largest execution time difference, with a 3× increase. Additionally, `freqmine` exhibits a 25% increase in execution time. This result highlights the importance of using calibrated models in simulation for accurate performance evaluation.




(a) Execution times (b) Memory vs C2C link bandwidth (c) Normalized C2C traffic types

Fig. 7: Comparison of (a) execution times, (b) main memory and C2C link bandwidth, and (c) normalized traffic types for different PARSEC programs across various gem5 architectures: reference single-chip, default chip-to-chip, and calibrated chip-to-chip.

Figure 7b shows the bandwidths for the C2C link and the main memory (combined for both local and remote main memory controllers) in the calibrated C2C architecture. We make two observations. First, the applications with the highest slowdown also show the highest C2C bandwidth utilization, with the slowdown being directly proportional to the bandwidth utilization. Second, the C2C link bandwidth is typically lower than the aggregated main memory bandwidth but varies depending on the application. E.g., `raytrace` shows C2C link bandwidth comparable to the main memory bandwidth, whereas `bodytrack` achieves only a third of the main memory bandwidth.

Figure 7c shows the normalized distribution of message types traversing the C2C link. We observe a similar distribution for the applications exhibiting the highest slowdown: `canneal`, `ferret`, `freqmine`, and `raytrace` benchmarks. However, the low proportion of snoop requests suggests that the slowdown is not caused by coherence traffic, but rather by the limited C2C link bandwidth. Thus, moving from the modeled PCIe-based interconnect to a more integrated chiplet-based interconnect could reduce the performance overheads observed in our experiments.

## VIII. Related Works

To our knowledge, this is the first work to propose a full-system software simulation of multi-chip designs connected using a cache-coherent chip-to-chip interconnect. However, we identify two areas of related work that are particularly relevant.

On the one hand, cycle-accurate NoC simulators such as BookSim2 [19], Noxim [20], Nostrum [21], and Garnet [22], originally designed for NoC simulations in monolithic chips, have been extended for multi-chiplet designs [23]. These extensions are useful for conducting simulations and examining various network topologies and routing algorithms for given traffic patterns. Our work is complementary, as we focus on providing the functionality to extend coherency across multiple coherent chips and thus produce realistic traffic that could be used by these NoC simulators.

On the other hand, cycle-accurate full-system simulators like gem5, with its Ruby CHI memory system, can accurately model monolithic many-core architectures [24]. Some methods coordinate multiple gem5 instances to simulate interconnected chips [25]–[27], but this setup requires each instance to run its own OS. Other approaches, such as OpenPiton [28], use a similar strategy but incorporate RTL for faster FPGA emulation. In contrast, our work uniquely enables the simulation of systems consisting of multiple coherent chips running under a single OS.

## IX. Conclusion

We introduce c2c-gem5, a new cache-coherent chip-to-chip interconnect model to accurately simulate multi-chip architectures in gem5. To extend single-chip coherence to the multi-chip system, we propose a new chip-to-chip interface controller, and minimal modifications to the CHI directory and message format to enable tracking of remote sharers and initiators. We use a real hardware platform to calibrate our model and run PARSEC workloads on two CHI chips operating under the same OS.

In future work, we will study the scalability of our model across many chips and improve the accuracy of the local NoC and chip-to-chip network by integrating our simulator with a NoC simulator, such as Garnet. We believe that c2c-gem5 takes an important step forward in enabling accurate simulation of cache-coherent multi-chip architectures and hope to inspire future work exploring optimizations in such architectures.

## X. Acknowledgments

This work was partially supported by ANRT-CIFRE Grant No. 2021/1201 and the European Processor Initiative (EPI) project, supported by the European High Performance Computing Joint Undertaking (JU) under Framework Partnership Agreement No. 800928 and Specific Grant Agreement No. 101036168 (EPI SGA2).

## REFERENCES


[1] S. Naffziger, N. Beck, T. Burd, K. Lepak, G. H. Loh, M. Subramony, and S. White, "Pioneering chiplet technology and design for the AMD EPYC and RYZEN processor families: Industrial product," in *Proceedings of the Annual International Symposium on Computer Architecture (ISCA)*, 2021, pp. 57–70.

[2] CCIX Consortium, "CCIX Consortium: Cache Coherent Interconnect for Accelerators (CCIX)," 2024, accessed: 2025-01-17. [Online]. Available: https://www.ccixconsortium.com/

[3] UCIe Consortium, "UCIe Consortium: Universal Chiplet Interconnect Express (UCIe)," 2024, accessed: 2025-01-17. [Online]. Available: https://www.uciexpress.org/

[4] J. Stuecheli, W. J. Starke, J. D. Irish, L. B. Arimilli, D. Dreps, B. Blaner, C. Wollbrink, and B. Allison, "Ibm power9 opens up a new era of acceleration enablement: Opencapi," *IBM Journal of Research and Development*, vol. 62, no. 4/5, pp. 8–1, 2018.

[5] C. Chen, J. Yin, Y. Peng, M. Palesi, W. Cao, L. Huang, A. K. Singh, H. Zhi, and X. Wang, "Design challenges of intra-and inter-chiplet interconnection," *IEEE Design and Test*, vol. 39, no. 6, pp. 99–109, 2022.

[6] J. Lowe-Power, A. M. Ahmad, A. Akram, M. Alian, R. Amslinger, M. Andreozzi, A. Armejach, N. Asmussen, B. Beckmann, S. Bharadwaj *et al.*, "The gem5 simulator: Version 20.0+," *arXiv preprint arXiv:2007.03152*, 2020.

[7] M. M. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood, "Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset," *ACM SIGARCH Computer Architecture News*, vol. 33, no. 4, pp. 92–99, 2005.

[8] ARM Limited, "Arm AMBA CHI architecture specification," accessed: 2025-01-17. [Online]. Available: https://developer.arm.com/documentation/ihi0050/latest/

[9] ——, "Arm N1SDP specifications," accessed: 2025-01-17. [Online]. Available: https://developer.arm.com/Tools%20and%20Software/Neoverse%20N1%20SDP

[10] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC benchmark suite: Characterization and architectural implications," in *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2008, pp. 72–81.

[11] L. Bertran Alvarez and D. Novo, "c2c-gem5: Full system simulation of cache-coherent chip-to-chip interconnects," 2025, accessed: 2025-01-17. [Online]. Available: https://gite.lirmm.fr/adac/c2c-gem5#

[12] ARM Limited, *Arm Neoverse N1 Core Technical Reference Manual*, https://developer.arm.com/documentation/100616/0401/, accessed: 2025-01-17. [Online]. Available: https://developer.arm.com/documentation/100616/0401/

[13] SiFive, Inc., "SiFive Performance P870 Core Series," 2024, accessed: 2025-01-17. [Online]. Available: https://www.sifive.com/cores/performance-p870d

[14] Semidynamics Technology Services, S.L., "Tensor unit technology," 2024, accessed: 2025-01-17. [Online]. Available: https://semidynamics.com/en/technology/tensor-unit

[15] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti *et al.*, "The gem5 simulator," *ACM SIGARCH computer architecture news*, vol. 39, no. 2, pp. 1–7, 2011.

[16] J. Randall, P. Benedicte, and T. Ta, "CHI-based Ruby protocol," https://github.com/gem5/gem5/commit/b13b4850951b4507cabee27a8c2, 2021, accessed: 2025-01-17.

[17] B. M. Beckmann and A. Gutierrez, "The AMD gem5 APU simulator: Modeling heterogeneous systems in gem5," in *Tutorial at the International Symposium on Microarchitecture (MICRO)*, 2015.

[18] A. Gutierrez, B. M. Beckmann, A. Dutu, J. Gross, M. LeBeane, J. Kalamatianos, O. Kayiran, M. Poremba, B. Potter, S. Puthoor *et al.*, "Lost in abstraction: Pitfalls of analyzing GPUs at the intermediate language level," in *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, 2018, pp. 608–619.

[19] N. Jiang, D. U. Becker, G. Michelogiannakis, J. Balfour, B. Towles, D. E. Shaw, J. Kim, and W. J. Dally, "A detailed and flexible cycle-accurate network-on-chip simulator," in *Proceedings of the International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2013, pp. 86–96.

[20] V. Catania, A. Mineo, S. Monteleone, M. Palesi, and D. Patti, "Noxim: An open, extensible and cycle-accurate network on chip simulator," in *Proceedings of the International Conference on Application-Specific Systems, Architectures and Processors (ASAP)*, 2015, pp. 162–163.

[21] Z. Lu, R. Thid, M. Millberg, E. Nilsson, and A. Jantsch, "NNSE: Nostrum network-on-chip simulation environment," in *Swedish System-on-Chip Conference (SSoCC)*, 2005.

[22] N. Agarwal, T. Krishna, L.-S. Peh, and N. K. Jha, "GARNET: A detailed on-chip network model inside a full-system simulator," in *Proceedings of the International symposium on performance analysis of systems and software (ISPASS)*, 2009, pp. 33–42.

[23] H. Zhi, X. Xu, W. Han, Z. Gao, X. Wang, M. Palesi, A. K. Singh, and L. Huang, "A methodology for simulating multi-chiplet systems using open-source simulators," in *Proceedings of the International Conference on Nanoscale Computing and Communication (NanoCom)*, 2021, pp. 1–6.

[24] A. Portero, C. Falquez, N. Ho, P. Petrakis, S. Nassyr, M. Marazakis, R. Dolbeau, J. A. N. Cifuentes, L. B. Alvarez, D. Pleiter *et al.*, "Compesce: A co-design approach for memory subsystem performance analysis in hpc many-cores," in *Proceedings of the International Conference on Architecture of Computing Systems (ARCS)*, 2023, pp. 105–119.

[25] A. Brokalakis, N. Tampouratzis, A. Nikitakis, I. Papaefstathiou, S. Andrianakis, D. Pau, E. Plebani, M. Paracchini, M. Marcon, I. Sourdis *et al.*, "Cossim: An open-source integrated solution to address the simulator gap for systems of systems," in *2018 21st Euromicro Conference on Digital System Design (DSD)*, 2018, pp. 115–120.

[26] A. Mohammad, U. Darbaz, G. Dozsa, S. Diestelhorst, D. Kim, and N. S. Kim, "dist-gem5: Distributed simulation of computer clusters," in *2017 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2017, pp. 153–162.

[27] F. Schätzle, C. Falquez, S. Heinen, N. Ho, A. Portero, E. Suarez, J. Van Den Boom, and S. Van Waasen, "Modeling methodology for multi-die chip design based on gem5/systemc co-simulation," in *Proceedings of the 16th Workshop on Rapid Simulation and Performance Evaluation for Design (RAPIDO)*, 2024, pp. 35–41.

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