



Modeling Chiplet-to-Chiplet (C2C) Communication for Chiplet-based Co-Design

Fabian Schätzle 


*Peter Grünberg Institute -
Integrated Computing Architectures
(ICA — PGI-4)
Forschungszentrum Jülich GmbH
Jülich, Germany*

Carlos Falquez 


*Jülich Supercomputing Centre,
Institute for Advanced Simulation
Forschungszentrum Jülich GmbH
Jülich, Germany*

Nam Ho 


*Jülich Supercomputing Centre,
Institute for Advanced Simulation
Forschungszentrum Jülich GmbH
Jülich, Germany*

André Zambanini 

*Peter Grünberg Institute -
Integrated Computing Architectures
(ICA — PGI-4)
Forschungszentrum Jülich GmbH
Jülich, Germany*

Johannes van den Boom 

*Peter Grünberg Institute -
Integrated Computing Architectures
(ICA — PGI-4)
Forschungszentrum Jülich GmbH
Jülich, Germany*

Estela Suarez  *[†]

*Jülich Supercomputing Centre,
Institute for Advanced Simulation
Forschungszentrum Jülich GmbH
Jülich, Germany*

* *SiPEARL, Maisons-Laffitte, France*

[†] *Computer Science Department,
University of Bonn, Bonn, Germany*

Abstract—Chiplet-based processor design, which combines small dies called chiplets to form a larger chip, enables scalable designs at economical costs. This trend has received high attention such that standards for chiplet design have rapidly established, including packaging, protocols, and Chiplet-to-Chiplet (C2C) interfaces. With numerous well-defined chiplet options available, hardware architects would leverage on the co-design process to make optimal decisions on design parameters.

An important performance limitation in multi-chiplet designs come from the protocol translation in the C2C communication, needed to maintain cache coherency and avoid risk of deadlocks. When integrating multiple chiplets, deadlocks can happen from both protocol and routing, making deadlock-free designs important.

This paper tackles these challenges by introducing a Chiplet-to-Chiplet Gateway (C2CG) architecture, a C2C interface that bridges two chiplet protocols and ensures deadlock-free C2C communication. We also extend the Coherent Hub Interface (CHI) protocol to support cache coherent data sharing among cores across chiplets. The complete design is implemented in the gem5 simulator, constructing a modeling tool for chiplet-based co-design targeting next-generation High-performance Computing (HPC) processors. We demonstrate the benefit of the model through a design space exploration of three 64-core Armv8 HPC processor configurations: monolithic, two- and four-chiplet. The exploration, using representative HPC benchmarks, provides insights into C2C parameters and studies the impact of Non-Uniform Memory Access (NUMA) configuration, giving valuable co-design feedback for hardware architects.

Index Terms—High-Performance Computing, Chiplet, Cache coherency

I. INTRODUCTION

As the limitations of Moore’s Law [1] become increasingly apparent, alternative approaches to chip design and manufacturing are gaining prominence. Among these, the chiplet-based architecture stands out as a promising solution to address the

challenges of scalability, cost, and performance in modern processors. Chiplets are smaller, modular integrated circuit blocks that can be combined into a single package, providing a flexible and scalable alternative to monolithic System-on-Chips (SoCs).

The increasing complexity and cost associated with the fabrication of large monolithic SoCs have made it challenging to achieve economic scalability. By partitioning these SoCs into smaller chiplets, manufacturers can optimize production processes, leveraging different semiconductor technologies for various components, thereby improving yield and reducing costs [2]. Furthermore, chiplets enable heterogeneous integration, allowing the combination of diverse functional blocks, such as CPU, GPUs, memory, and other specialized accelerators, each manufactured using the most suitable process technology [3]. This flexibility is crucial for meeting the diverse and evolving demands of modern applications, which range from high-performance data centers [4] to energy-efficient edge devices [5]. Moreover, chiplets enable accelerated innovation cycles, as individual chiplets can be reused in new system designs. This approach reduces time-to-market and fosters a dynamic and competitive semiconductor landscape.

Chiplet-based designs leveraging on 2.5D packaging technology have now matured beyond the early developmental stage. Successful examples of chiplet-based CPU designs include AMD EPYC and Ryzen [2], Graviton3/4 [6]–[8], Kunpeng 920 [9], and Intel Sapphire Rapids [10].

The two most established approaches of 2.5D packaging for chiplet interconnection are standard packaging using organic substrates and advanced packaging using silicon interposers [11], [12]. Standard packaging provides limited bandwidth with bump pitches of approximately 100-130 μm ,

suitable for low bandwidth applications. Advanced packaging, on the other hand, is designed for high bandwidth requirements, utilizing micro bump pitches ($25\text{--}55\mu\text{m}$), but comes with a higher cost. At the physical layer, C2C interconnects are implemented either through serial or parallel links over the packaging layer, connecting to physical interfaces in sending and receiving chiplets. While requiring fewer connections supporting longer distances, serial interfaces [13] incur more power consumption and latency. In contrast, parallel interfaces [14] have lower latency and power consumption but require more connection links [12].

At the protocol layer, the C2C interface translates the chiplet protocol into the physical link protocol for transfers over the C2C-Link, and vice versa. In addition, the protocol layer also manages C2C cache-coherent communication for data sharing between cores across chiplets [11], [15]. Arm has recently promoted the ecosystem for chiplet designs and introduced the Advanced Microcontroller Bus Architecture 5 (AMBA5) CHI C2C protocol [16]. Arm recommends using an interface structure called Coherent Multichip Link (CML) Gateway [17], which acts as a proxy for CHI message transfers, maintaining cache coherence across chiplets.

Several options exist for chiplet-based designs, so that selecting appropriate design parameters for specific applications, especially for HPC processors, remains highly challenging. A co-design process that provides a comprehensive view of the pros and cons of various design options is necessary to guide chip architects in making optimal decisions [18]. In the co-design process, the uses of modeling tools and benchmarking are very important for architectural exploration, verification, and performance tuning. For example, modeling tools played a key role in the success of the A64FX processor co-design [19].

This paper presents a gem5 model for co-design analysis of C2C communication, with a focus on AMBA5 CHI chiplet-based architectures. We introduce a reference implementation of a C2C interface, called C2CG, inspired by the CHI C2C specification. The C2CG supports key requirements for C2C communication: It converts the on-chip CHI packets from the chiplet into C2C-Link containers for C2C transfers, and handles C2C transactions correctly. In addition, the C2CG is carefully designed to avoid deadlock issues within chiplet-based architecture [20]. The model also extends the AMBA5 CHI protocol to maintain C2C cache coherence through a customized coherence directory in the CHI Home Node (HN), enabling multiple multi-core chiplets to share data in a cache coherent manner. Using the developed model, we perform a co-design analysis of two and four chiplet designs by examining key C2C parameters for the standard and the advanced packaging and compare both designs with a monolithic design.

The contributions of our paper are:

- We extend the CHI protocol with C2C cache coherence support including design consideration of C2C interface layers to address the design challenges for scalable, high bandwidth, and low latency cache coherent C2C interconnects (Section III).

- We examine the risk of routing deadlock issues in chiplet interconnects, and present how we ensure deadlock avoidance in our design (Section III-B).
- We present a detailed C2C communication model using a widely-used gem5 simulator [21], including an implementation of the C2CG structure and the extension of the CHI protocol (Section IV).
- We present the advantages of using the C2C model through a case study evaluating three designs for 64-core Arm Neoverse processors: monolithic, two chiplets, and four chiplets (Section V). The model supports the full-system simulation (FS) mode and provides valuable co-design feedback, e.g., suggesting the optimal C2C configuration using representative HPC benchmarks.

II. BACKGROUND & RELATED WORK

Chiplet-based designs relying on 2.5D chiplet integration are being successfully adopted to manufacture recent high-end processors [2], [6]–[10]. The transition is primarily motivated by (1) the ever-increasing die sizes, hitting the reticle limit; (2) the low yield and therefore higher cost of large monolithic chips; and (3) the added scalability and flexibility to build heterogeneous chips, which chiplet designs provide and monolithic designs do not.

To enable the integration of chiplets from different vendors, standardized packaging, interfaces and protocols are needed [11], [12]. Packaging choices are the use of organic package substrates or silicon interposers. From both the latter (silicon interposers), which use micro-bump pitches ($25 - 55\mu\text{m}$), provide higher bandwidth supporting up to 64 lanes per cluster for wire width [22]. While package substrates and passive interposers only have wires, the advanced packaging of active interposers allows transistors to be embedded, enabling the designs of Network on Interposer, increasing the scalability of chiplet integration [23]–[25]. The standard of serial and parallel interfaces, along with the protocols for C2C-Link transmission, are also well established [13], [14], [22].

While the standards for physical integration have been well defined, there are still concerns at the system architectural level. One of the challenges is handling coherent data sharing between cores across chiplets. Some efforts have been dedicated to that, including CCIX [26], Universal Chiplet Interconnect Express (UCIe) [22], and the recent release of the AMBA5 CHI C2C protocol by Arm [16], [17]. Furthermore, when connecting multiple chiplets where each chiplet implements a Network-on-Chip (NoC) that is deadlock-free, the chiplet architecture can still suffer from routing deadlocks. Thus, the deadlock-free design for chiplet integration must be carefully addressed. Notable proposals include increasing virtual channels [27], using turn-restriction routing [28], or applying in-transit buffers at the boundary chiplet router [20].

Pre-silicon modeling tools are critical for chiplet-based co-design, however, we notice only a few example in the literature [24], [29]–[31]. None of them are specially dedicated to C2C cache coherence protocol analysis or investigate the deadlock-free design aspect. In [32], the cache coherent

chiplets only add latency to C2C links without accounting for physical parameters of the C2C controllers. Furthermore, all chiplets are modeled by one single network, which is not modular. It also assumes that every chiplet knows the other chiplets components which is not realistic when interfacing with other vendor chiplets. Our gem5 analysis model addresses the co-design challenges for chiplet-based processors with a focus on emerging CHI C2C cache coherence and the design for deadlock-free. In addition, we provide a detailed exploration of the physical parameters of the chiplet interface, thus providing valuable co-design feedback.

III. CHIPLET-BASED ARCHITECTURE WITH AMBA5 CHI C2C

We discuss in this section the requirements for a chiplet-based processor where a monolithic compute chip is split into multiple smaller compute chiplets, connected by a fully coherent CHI C2C interconnect [33], [34].

Figure 1 shows three 64-core architectures by monolithic (*R1*, Figure 1(a)), two chiplets (*R2*, Figure 1(b)) and a four chiplet (*R3*, Figure 1(c)) design, inspired by the reference Arm Neoverse design [33]. The purple area represents the memories, the light blue corresponds to the compute chiplets, and the dark blue to the C2C interfaces. The chiplet interconnect in *R2* and *R3* uses point-to-point topology, either through a packaging substrate or a passive silicon interposer. Furthermore, we assume four NUMA regions for each design, where cores within each NUMA allocate and access resources from their nearest memory.

Figure 1(d) shows the hardware layout within the NUMA0 region (e.g., in *R2*), connected as a 4×4 2D-Mesh, and implementing X-Y routing and the CHI protocol, where each *Request*, *Snoop*, *Data*, and *Response* message type uses a dedicated Virtual Channel (VNET). We placed the nodes according to our previous research [35]. The Request Node (RN) consists of one or more processor cores with caches. The HNs acts as a coherence directory and optionally holds a slice of System-Level-Cache (SLC). Each HN is responsible for a unique memory address range. The Subordinate Node (SN) is the interface to memory.

Let us define the CHI routing transfers within these architectures as follows:

Definition 1. *Outbound packet* is a routing packet of any CHI message type where the source and destination are not within the same chiplet. When the outbound packet is injected in the destination chiplet, it is referred to as **inbound packet**.

Definition 2. *Intra packet* is a routing packet where the source and destination are within the same chiplet.

A. C2C cache coherence

CHI implements the MOESI cache coherency states: Unique Dirty (UD, M), Shared Dirty (SD, O), Unique Clean (UC, E), Shared Clean (SC, S), Invalid (I, I). Originally, CHI was designed for on-chip cache coherence. The C2CG node shown in Figure 1 and Figure 2 bridges two chiplet CHI protocols. It

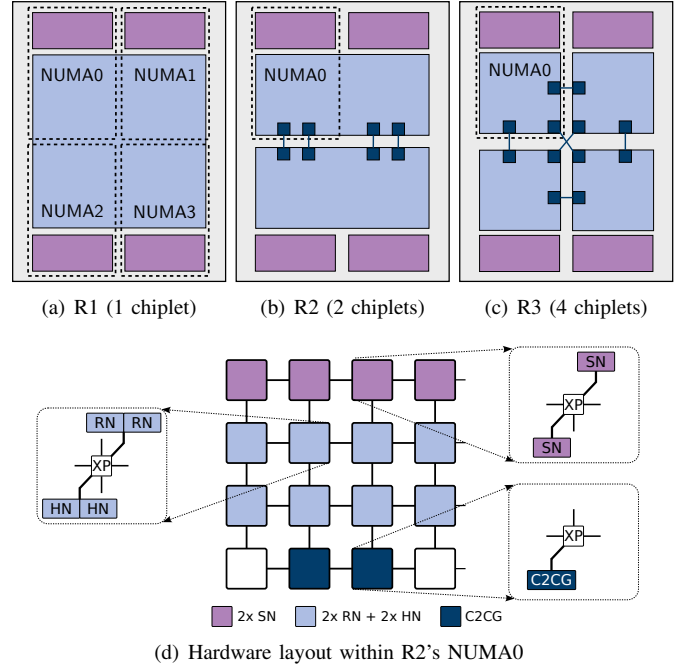


Fig. 1. Chip designs with 1, 2 and 4 compute chiplets (light blue) attached to their HBM2 (purple), which is separated into four NUMA regions. The dark blue boxes are the C2C interfaces. We call the configurations with 1, 2, and 4 compute chiplets (a) *R1*, (b) *R2* and (c) *R3*, respectively. Diagram (d) shows an example hardware layout for *R2* in NUMA0 and crosspoints (XP).

acts as a RN or HN proxy and handles the exchange of cache coherence messages across chiplets.

The C2CG within a chiplet works as a hub for all outbound packets forwarded by the local CHI components. The advantage of the C2CG is that the CHI components within a chiplet do not need to be aware of the NoC details in other chiplets. Any outbound CHI packet is sent to the corresponding C2CG, which then packs the packets into containers and sends them over the C2C-Link to another chiplet. This satisfies the IP protection required when integrating chiplet IPs from different vendors, avoiding disclosure of global information.

B. C2C deadlock avoidance

Within a chiplet, the transfer of the four CHI message types *Request*, *Snoop*, *Data*, and *Response* uses separated VNETs to prevent protocol deadlocks. For cross-chiplet transfers, the C2CG uses dedicated buffers for each message type to prevent protocol deadlocks as well. In addition, the NoC design ensures deadlock-free via X-Y routing. However, connecting multiple deadlock-free chiplets can potentially lead to routing deadlocks.

We proposed a deadlock-free routing solution by adapting the concept of In-Transit Buffer (ITB) [36] for C2C interconnect. Our solution is similar to the Remote Control (RC) design [20]. The RC design considers a small outbound packet buffer size of 1 and requires chiplet modification for a permission network for atomic buffer allocation for deadlock avoidance. In contrast, we eliminate the need for a permission

network by featuring within the C2CG an optimal transaction buffer size to prevent routing deadlocks. When the C2CG can not accept outbound packets (i.e., the transaction buffer is full), the C2CG will drop the packet and send a *RetryAck* packet back to the sender. The sender will resend the outbound packet in the future once it receives a *PCrdGrant* notification message from C2CG, see Section IV-B. We provide formal proof in the Appendix showing that this mechanism effectively avoids routing deadlocks.

IV. MODELING C2C COMMUNICATION

Figure 2 shows the exemplary two-chiplet processor architecture that we modeled in gem5. The Processing Units (PUs) and memories are modeled with gem5’s classic memory system. Caches, interconnect network, C2C-Link, and the CHI nodes (HN, SN and C2CG) are implemented in gem5’s ruby system supporting cache coherence. The CPU core, which is a RN in CHI, consists of PU, separate L1 instruction and data cache, as well as a unified L2 cache. The current version of gem5 supports CHI cache coherence only for monolithic chip architecture. To enable C2C architecture modeling, we extended the CHI implementation in gem5 by adding C2CG components and C2C-Links to mimic physical links.

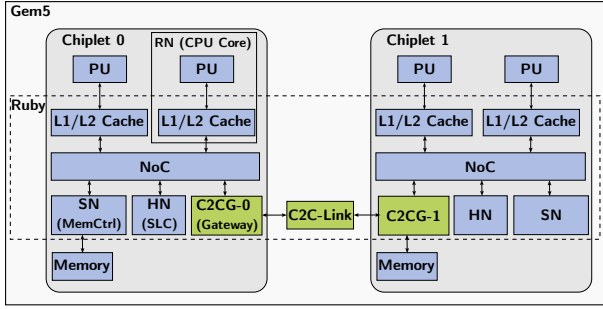


Fig. 2. Gem5 model with 2 chiplets, each containing 2 RNs (PU with L1 and L2 caches). These are connected to the NoC, from which they can access the HN (SLC), the SN (memory controllers), and the C2CG (highlighted). The C2CG are interconnected via the C2C-Link.

A. C2CG’s architecture

Figure 3 shows the modeled C2CG structure. It connects to the Network Interface (NI) of the NoC and to the C2C-Links. Furthermore, we implemented a credit-based C2C-Link to model the physical layer of the C2C interface, providing bandwidth and clock frequency flexibility. The C2CG is responsible for: (1) packing CHI outbound packets into C2C-Link containers to be sent over C2C-Links to reach the C2CG of other chiplets, and vice versa; (2) serving cache coherence transactions among the chiplets; (3) preventing both protocol and routing deadlocks across chiplets via dedicated input/output buffers (**ReqIn/Out**, **RspIn/Out**, **DatIn/Out**, **Snpln/Out**) for each CHI message type (*Request*, *Response*, *Data*, *Snoop*) and the *Retry* mechanism.

The C2CG uses a *Transaction Table* to track pending outbound transactions (e.g., *Request* and *Snoop*) in order to return responses (*Response*, *Data*) from neighbouring

chiplets to the local initial requestor. For example, within a chiplet, when a local RN issues a memory read request with a memory address belonging to a remote HN in another chiplet, an outbound read request packet is routed directly to the corresponding local C2CG¹. The C2CG stores the tracking information for this outbound packet in the *Transaction Table*. When the remote HN receives the read request, it will perform a cache coherence transaction and reply to the initial RN with response data (Section IV-C). Once receiving the response data from the neighbouring C2CG, the C2CG looks up the pending transaction in the *Transaction Table*, prepares an inbound response packet, and sends it back the local RN.

B. C2C transfers

We detail in Figure 3 the flow of C2C transfers² through an example involving two C2CGs for the two-chiplet architecture shown in Figure 2.

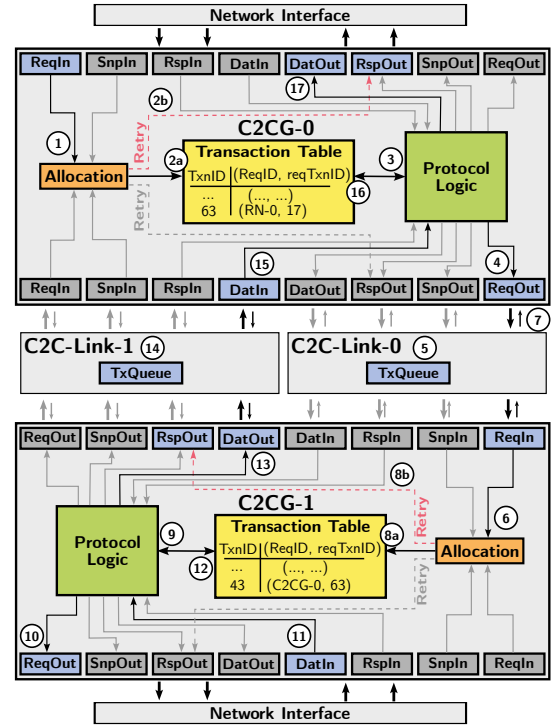


Fig. 3. C2C transfers between two chiplets through two C2CGs and C2C-Links. C2CG’s architecture has a *Transaction Table*, *Allocation* and *Protocol Logic* to correctly handle transactions between two chiplets.

1) *Request transfers*: When there is an incoming outbound request (e.g., *Request*) in the **ReqIn** buffer of the C2CG-0 within chiplet 0, the *Allocation* is triggered ① to allocate a tracking slot in the *Transaction Table* including a needed buffer to handle transactions for this request ②a. If no resources are available, a *RetryAck* message is generated and placed in the **RspOut** buffer to inform the requestor that the request

¹Given an address, the routing destination is determined through an address hash function.

²Throughout the description, we imply that transfers occur in packet units.

is temporarily rejected ②³. When C2CG-0 has resources available, it reserves a slot in the *Transaction Table* and notifies the rejected requestor by sending a *PCrdGrant* message. The resent request thus will be accepted when reaching C2CG-0.

Once a slot in the *Transaction Table* is allocated, a C2CG-0 to C2CG-1 transaction is created (*TxnID*=63). And, C2CG-0 extracts the outbound request and stores in this slot the requestor node (*ReqID*=RN-0), and the requestors transaction ID (*reqTxnID*=17). This information is used to construct the inbound response returned to the requestor when receiving an outbound response from the neighboring C2CG-1. The *Protocol Logic* then constructs a new outbound request with updated IDs (*ReqID*=C2CG-0 and *TxnID*=63) ③ before packing it into a container and sending over the C2C-Link to C2CG-1.

The communication between two C2CGs over C2C-Links is credit-based, following the CHI flow control mechanism. C2CG-0 must obtain a granted credit from the C2C-Link before sending the outbound request. Upon granted a credit, C2CG-0 places the request in the **ReqOut** buffer ④, from there, it will be shifted to the *TxQueue* of C2C-Link-0 ⑤ for a transmission to the **ReqIn** of C2CG-1. Note that once the request message is dequeued from **ReqIn**, C2C-Link-0 sends credits back to C2CG-0 to inform that the input buffer of C2CG-1 has free space ⑦.

Upon receiving an incoming outbound request in the **ReqIn**, C2CG-1 allocates a tracking slot in its *Transaction Table* ⑥. If resources are not available, C2CG-1 applies the *Retry* mechanism as well, by rejecting the request and sending a *PCrdGrant* message back to the requestor in C2CG-0. Thus, in C2CG-1, once a tracking slot is allocated (*TxnID*=43) ⑧, the *Protocol Logic* constructs an inbound request from the received outbound request, and forwards it to the **ReqOut** buffer ⑩, which is then injected into the NoC of chiplet 2.

2) *Response transfers*: In chiplet 2, the destination (e.g., HN) responds to the inbound request with a CHI message type *Response*, either with or without data (*Data*). We describe the flow of *Data* response, where transfers utilize dedicated **DatIn** and **DatOut** buffers. The flows of other response types are similar, except using different input and output buffers.

The *Protocol Logic* in C2CG-1 is notified upon receiving an outbound *Data* response in **DatIn** ⑪. The *Protocol Logic* looks up in the *Transaction Table* for the *TxnID*=43 encoded in the received *Data* response packet ⑫. Using the found tracking slot, *ReqID*=C2CG-0 and *reqTxnID*=63 are extracted to construct a new outbound *Data* response, which C2CG-1 sends to C2CG-0. The outbound response is forwarded to the **DatOut** buffer ⑬, then stored in the *TxQueue* of C2C-Link-1 ⑭, finally reaches C2CG-0 in **DatIn** buffer. At C2CG-0, the outbound response is handled similarly: the *Protocol Logic* ⑮ searches in the *Transaction Table* ⑯ for the pending transaction with *TxnID*=63, constructing an inbound *Data* response from the tracking slot and forwarding it to the

DatOut buffer. The inbound response will be then injected in the NoC of chiplet 0 ⑰, and finally reach the initial requestor.

C. Maintaining C2C cache coherence

1) *CHI on-chip cache coherence*: CHI was originally targeted for NoC architectures, implementing the MOESI cache coherence. In a monolithic architecture (Figure 1(a)), each HN in the CHI system includes a directory that tracks who shares the memory address range within the chiplet. A RN sends requests always to the HN, which acts as a point-of-coherency. The HN lookups its directory, and in case there are sharers, a *Snoop* is broadcasted to keep their cache state in coherency, and the sharer will respond data to the RN. If the HN has data within its SLC, a *Response* is sent directly to the RN. Otherwise, the request is forwarded towards memory (SN), which then responds to the RN via the HN. In both cases the HN updates the sharers in its directory after the transaction is completed.

2) *CHI C2C cache coherence*: We extended the directory in HNs to track not only on-chip CHI sharers but also the off-chip sharers along with their corresponding on-chip C2CG to maintain C2C cache coherence. Figure 4 illustrates transactions for maintaining cache coherency across chiplets.

Assume that RN-0 within chiplet 0 issues an outbound memory read request (*ReadShared*) for an address mapped to a memory device located in chiplet 1 ①. Once this request (*ReqID*=RN-0) arrives at C2CG-0, it is extended by a remote requestor field (*RmtReqID*). C2CG-0 fills in *RmtReqID*=RN-0 and also alters *ReqID*=C2CG-0 ②. When the request reaches C2CG-1, the *ReqID* is updated again to *ReqID*=C2CG-1 ③. Upon reaching HN-1, a data read to the memory device (SN-1) is issued. Then, HN-1 updates its directory with the new remote sharer and sharing state accordingly (State = UC and Sharer = RN-0 (C2CG-1), and sends a *Data* response back to the RN-0 ④.

Subsequently, upon RN-1's retrieval of the same data, HN-1 identifies the presence of an existing sharer, RN-0. Then, HN-1 constructs an outbound *Snoop* request (*SnpShared*) with the remote destination field encoded as *RmtDest*=RN-0 ⑤. This *Snoop* request is sent towards RN-0 via C2CG-1 and C2CG-0. When RN-0 receives the *Snoop* request, it responds to the HN-1 with an outbound snoop *Response* ⑥. HN-1 uses the *RmtRespID* to assign the snoop *Response* to the sharer. This is mandatory for the directory update when there are multiple remote sharers which are accessed via the same C2CG. Once HN-1 receives all responses for the *Snoop* request, it forwards *Data* to the requestor RN-1 and updates its directory ⑦.

V. EVALUATION

We leveraged the gem5 simulation framework [37] developed in a previous work for studying modern Arm Neoverse architectures [35], and extended it to support the C2C communication described above. Our enhanced model supports FS mode, enabling us to evaluate the impact of C2C communication and provide valuable co-design feedback. In this section, we present a case study evaluating three 64-core processor

³C2CG queues the rejected request ID so that the rejected requestor is informed about resending the packet later.

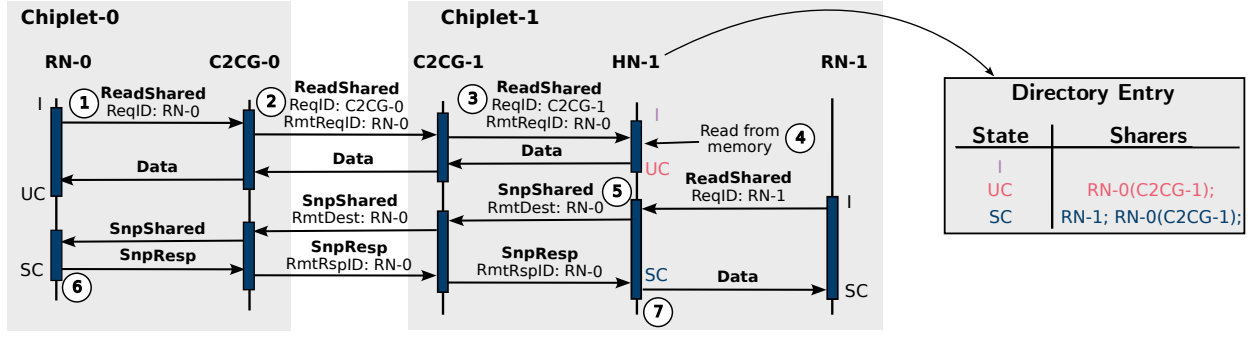


Fig. 4. The C2C transaction flow showing the updated message fields and HN directory modifications to ensure cache coherence across chiplets. The colored directory entries demonstrate the updated *State* and *Sharers* taking place within the HN after each transaction. Next to the RNs their cache states are shown.

designs shown in Figure 1: monolithic ($R1$), two chiplets ($R2$), and four chiplets ($R3$). For all three architectures, we consider NUMA support and assign SLC slices per NUMA node.

Table I details the evaluated architecture parameters. The core architecture implements Armv8 using $2 \times$ SVE engines. The chiplet NoC is modeled using the SimpleNetwork gem5 interconnection model with X-Y routing, and implements the AMBA5 CHI protocol with C2C cache coherence. Transaction buffers (#TXN) at L1, L2, and SLC caches are empirically explored to analyse which sizes are sufficient to track in-flight memory requests, maximizing memory bandwidth utilization for HBM2. We evaluate C2C parameters for two-/four-chiplet designs as follows: the number of C2CGs (#C2CGs), *Transaction Table* size (#TXN)⁴, *TxQueue* size(TXQ), and a C2C-Link bandwidth range aligned with the UCle specification (C2C-Link-BW).

We ran five representative HPC benchmarks in FS mode: three memory bandwidth bound kernels (STREAM-Triad [38], miniFE [39], waLBerla [40]), and two memory latency bound graph kernels Breadth-first Search (BFS) and Single-source shortest path (SSSP) from the Ligra benchmark suite [41]. The problem size was configured, e.g., exceeding the capacity of the SLC cache, to ensure meaningful assessments. We used the STREAM benchmark to study memory bandwidth utilization, and a microbenchmark (ietubench [42]) to characterize core-to-core latency.

A. Characterizing memory bandwidth

We ran STREAM-Triad ($a \leftarrow C \cdot b + c$) by allocating all vectors to memory located in either NUMA0 or NUMA1⁵, and pinning the execution to all cores in NUMA2 to examine intra- and C2C traffic. Figure 5 plots the breakdown of intra- and C2C traffic. For all three architectures, *Snoop* traffic is very low since there is no data-sharing in STREAM-Triad. Compared to $R1$, traffic in $R2$ and $R3$ increase by 80%, involving both intra- and C2C traffic due to additional hops traversing over C2CGs. Thus, maximizing memory bandwidth

⁴We model the *Transaction Table* by using Transaction Buffer Entries (TBEs) in gem5.

⁵This was done using the 'numactl' utility, supported in our gem5 model.

TABLE I
ARCHITECTURE PARAMETERS FOR SIMULATIONS. THE NUMBER OF CORES, MEMORY CHANNELS, C2CGs, MESH SIZE AND MEMORIES ARE LISTED PER CHIPLET.

Design	R1	R2	R3
Clocks	System: 1.6 GHz; Core: 2.6 GHz; NoC: 2.0 GHz;		
Core	Armv8 O3; SVE: 2x256 bit; L1I: 64 KiB, 4-way, 1 cycle hit latency, #TXN: 96 L1D: 64 KiB, 4-way, 2 cycle hit latency, #TXN: 96 L2: 512 KiB, 8-way, 4 cycle hit latency, #TXN: 96		
	#Cores: 64	#Cores: 32	#Cores: 16
NoC	Protocol: CHI; Model: SimpleNetwork; Routing: XY; Router/Link latency: 1 cycle; #VNETs: 4; VNET credits: 8 Link configuration: 2 physical channels per VNET; Shared L3: 1MiB, 16-way, 10 cycle hit latency #TXN: 128		
	Mesh: 8x8	Mesh: 4x8	Mesh: 4x4
C2C	N/A	#C2CGs:{2, 4}; #TXN:{128,256,512}; allocation/retry/stall latency: 1 cycle; message processing latency: 10 cycle; Credits per VNET: 8 C2C-Link-BW(GB/s):{64, 128, 256, 512}; TXQ(Byte):{128, 256};	#C2CGs:{3, 6}
Memory	Model: HBM2; Bandwidth per channel: 38.4 GB/s		
	#Channels: 32 Size: 4 GB	#Channels: 16 Size: 2 GB	#Channels: 8 Size: 1 GB

utilization requires optimizing C2C configurations to mitigate the impact of C2C traffic.

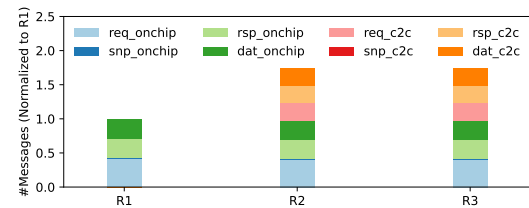


Fig. 5. On-Chip and C2C traffic measured with STREAM-Triad. $R1$ is the monolithic configuration, while $R2$ and $R3$ are built with two and four compute chiplets, respectively. For three architectures, vectors are allocated in NUMA0, and executions are pinned to all cores in NUMA2.

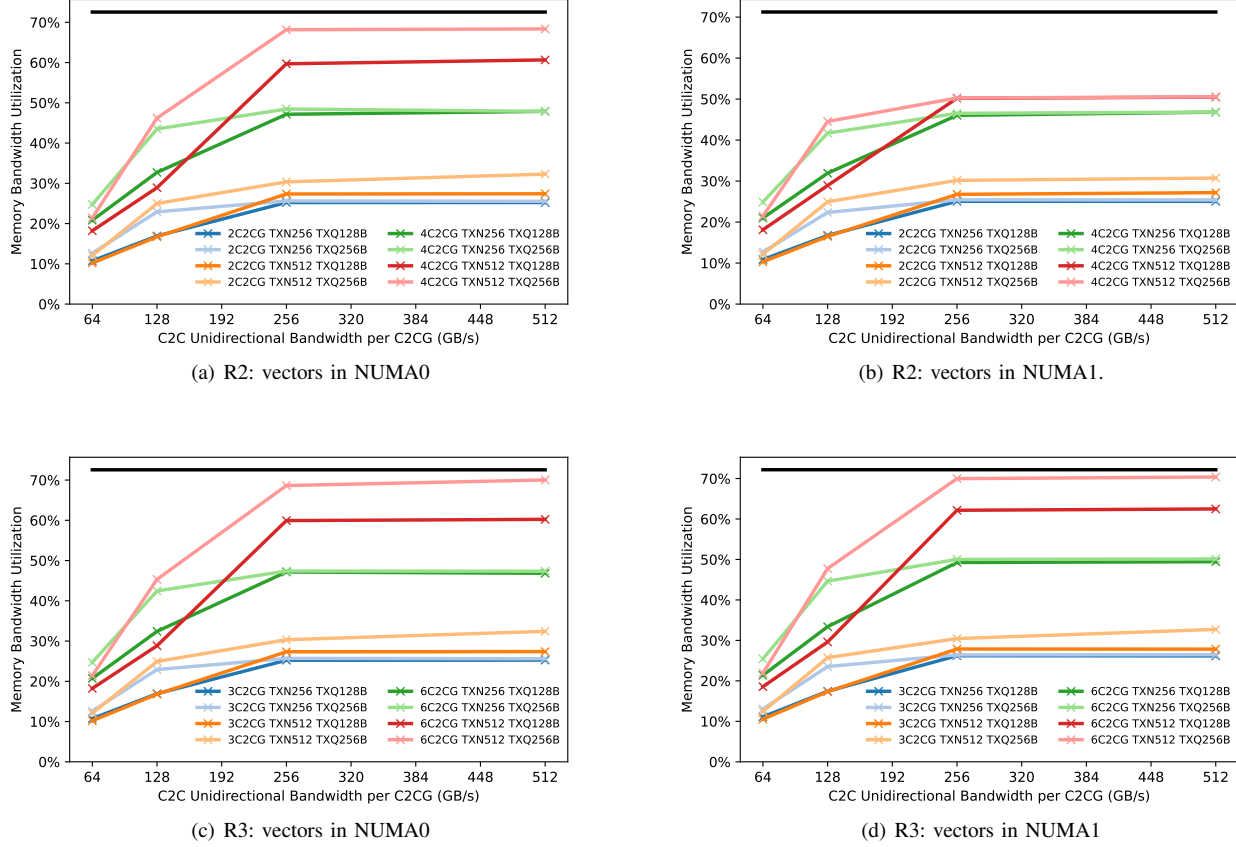


Fig. 6. Memory bandwidth utilization for STREAM Triad with active processors in NUMA2 and vectors allocated in different NUMA regions. The black horizontal line indicates the memory bandwidth for the R1 chip. **Optimal parameters** suggested for **R2**: $4 \times \text{C2CG}$, $\# \text{TXN}=512$, $\text{TXQ}=256\text{B}$, $\text{C2C-Link-BW}=256 \text{ GB/s}$; **R3**: $6 \times \text{C2CG}$, $\# \text{TXN}=512$, $\text{TXQ}=256\text{B}$, $\text{C2C-Link-BW}=256 \text{ GB/s}$.

We investigated optimized C2C parameters for *R2* and *R3* to maximize memory bandwidth utilization for NUMA configurations as shown in Figure 6. The results are explored for the range of C2C parameters in Table I, using STREAM-Triad, conducted under different NUMA execution scenarios: Execution was pinned to all cores in NUMA2, while the vectors were allocated in either NUMA0 or NUMA1.

For all NUMA execution scenarios, the monolithic architecture (*R1*) achieves bandwidth utilization around 71% of the peak. For the two-chiplet architecture (*R2*), using $4 \times \text{C2CG}$ s with a C2C-Link bandwidth of 256 GB/s/C2CG, along with relevant sizes for *Transaction Table* and *TxQueue* ($\# \text{TXN}=512, \text{TXQ}=256 \text{ B}$, respectively) are desired (Figure 6(a) and Figure 6(b)). While vector allocation on NUMA0 achieves a good result, vector allocation on NUMA1 saturates bandwidth at around 50%. This happens due to the X-Y routing bottleneck at the C2CG location (Section V-B).

For the four-chiplet architecture (*R3*), using $6 \times \text{C2CG}$ s to provide two connections to each chiplet is sufficient (Figure 6(c) and Figure 6(d)). The point-to-point C2CG connection in this architecture shows good performance in both NUMA execution scenarios.

The optimal parameters explored for *R2* and *R3* in this section are mostly used for the discussion in the following sections.

B. The impact of X-Y routing

We investigate in this section why, in *R2*, bandwidth saturates for vector allocation in NUMA1 and execution in NUMA2, even using additional C2CG as observed in the previous section. Figure 7 shows heatmaps of the average stall time for *R1*, *R2*, and *R3*, with the optimal parameters explored for *R2* and *R3* (Section V-A), using STREAM-Triad. Both *R1* and *R3* show lower stall times compared to *R2*. In *R1*, routing packets from cores in NUMA2 follow X and then Y directions to reach 8 memory controllers in NUMA1, suffering the lowest stall times due to the large 8×8 NoC. In *R3*, routing packets from NUMA2 chiplet to NUMA1 chiplet are redirected to the C2CG located at the router coordinates (3,3) and (3,2) in both chiplets. Once reaching the NUMA1 chiplet, the packets traverse the X and Y paths to the upper memory controllers without suffering from a bottleneck.

However, in *R2*, all routing packets from the NUMA2 chiplet have to first arrive at the C2CG located at routers (1,3), (2,3) in the upper chiplet (NUMA0). Later on, as packets

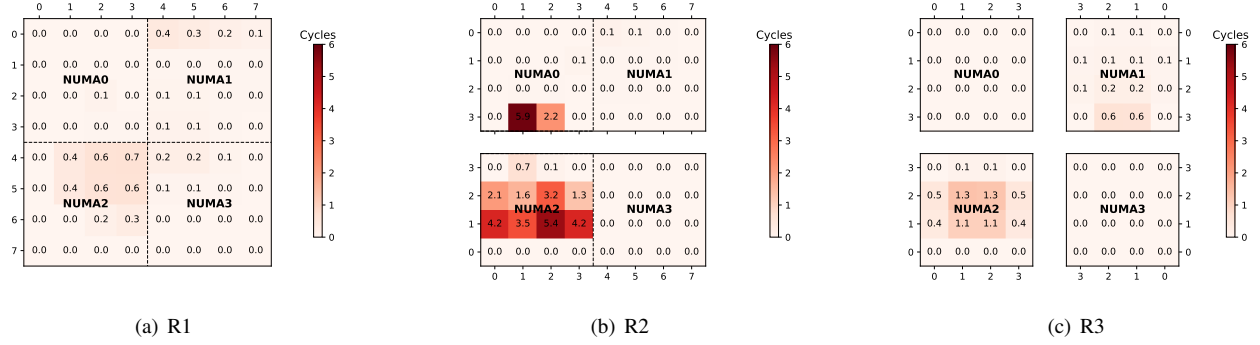


Fig. 7. Heatmaps of average stall times (cycles) of *R1* (8×8 NoC), *R2* (two 4×8 NoCs) and *R3* (four 4×4 NoCs) running with STREAM-Triad when vectors are allocated in NUMA1 and executions are on NUMA2. X and Y axes show the router coordinates. *R2*, and *R3* are configured with optimal parameters explored in Figure 6. The C2CGs positions for both chiplets in *R2* are at router coordinates (1, 3), (2, 3), (5, 3) and (6, 3); in *R3* for all chiplets: (0, 2), (0, 3) (1, 3), (2, 3), (3, 3), (3, 2).

traverse on the X and then Y direction to reach the memory controllers in the NUMA1 chiplet, they suffer congestion.

C. The impact of RetryAck messages

The implementation of the *Retry* mechanism within the C2CG structure ensures deadlock-free routing. However, this approach has a performance drawback: when the *Transaction Table* resources are not available to accept an outbound request, the sender is required to resend the packet, potentially impacting performance. We show in Figure 8 the effect of reducing the *RetryAck* messages when scaling the size of *Transaction Table* for architectures *R2* and *R3*, when vectors are allocated either in NUMA0 (Figure 8(a)) or in NUMA1 (Figure 8(b)) and executions are pinned to all cores in NUMA2. The observed trend indicates that when using sufficient *Transaction Table* entries, the number of *RetryAck* decreases, resulting in higher speedup gains. In *R2* ($4 \times$ C2CGs) and *R3* ($6 \times$ C2CGs), speedup saturates at a *Transaction Table* size of 384 entries when allocating vectors in NUMA0. Meanwhile, the best performance is achieved for a *Transaction Table* with 512 entries when allocating vectors in NUMA1.

D. Characterizing core-2-core latency

We used the customized core-to-core latency microbenchmark [42] to measure the average latency of cache coherence protocol messages exchanged between two cores. The measurement is performed by using one core to update shared data while the other observes the change.

Figure 9 plots the intra- and C2C traffic between core 0 and core 63 for three architectures. Both *R2* and *R3*, configured with the optimal parameters explored for maximizing bandwidth, show increased traffic with significant rises for both intra and C2C *Snoops*.

The heatmap of core-to-core latency for each architecture is shown in Figure 10. The measurements are between core 0 (NUMA0) and either core 15 (NUMA0), core 31 (NUMA1), core 47 (NUMA2), or core 63 (NUMA3). In *R1*, the cross-NUMA latencies range between 155 ns and

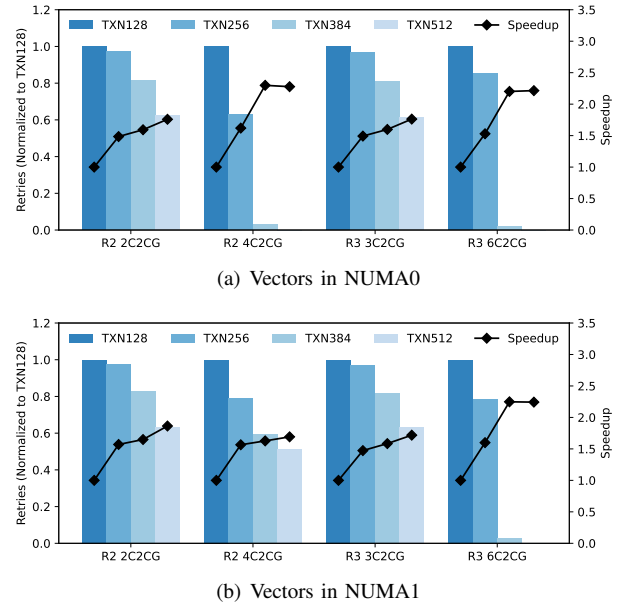


Fig. 8. Comparison between retries and speedup for *R2* and *R3* architectures by executing the STREAM-Triad on NUMA2, and allocating vectors either in NUMA0 or in NUMA1. Retries are normalized to TXN128 of each group.

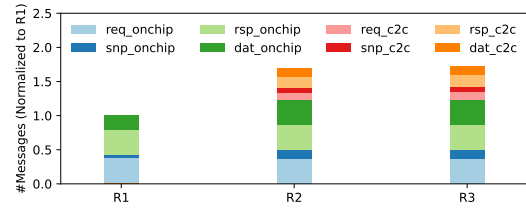


Fig. 9. On-Chip and C2C traffic are measured between core 0 and core 63. *R2*, and *R3* are configured with optimal parameters explored in Fig. 6.

181 ns with insignificant deviation. This suggests that the NUMA configuration support for this architecture may not be necessary. That means that SLC accesses can be uniformly distributed across all SLC slices within the chip, as seen

in the design of Graviton3, which aims for a large SLC capacity [43]. In contrast, cross-chiplet latencies within *R2* and *R3* show significant latency increases, up to 330 ns. Thus, NUMA configuration should be supported in these chiplet architectures, as it is critical for performance optimization in applications sensitive to latency. The approach is also applied in the chiplet-based design of AMD EPYC [2].

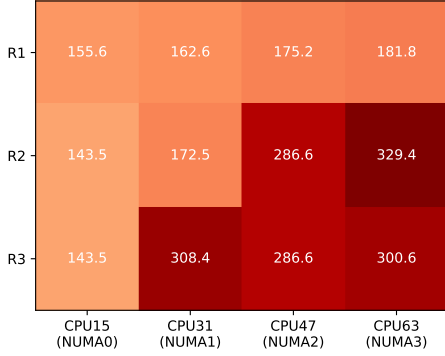


Fig. 10. Core-to-core latency (ns) are measured between core 0 located in NUMA0 and either core 15 (NUMA1), core 31 (NUMA1), or core 63 (NUMA2). *R2*, and *R3* are configured with optimal parameters explored in Figure 6.

E. Performance evaluation

We evaluate the performance of three architectures, where *R2* and *R3* are configured using the optimal parameters explored in Section V-A (Figure 6). The evaluations are done with two graph processing kernels Ligra-BFS and Ligra-SSSP⁶, the SpMV kernel from miniFE, and a stencil kernel from waLBerla known as UniformGridBenchmark. All codes (except the Ligra ones) are OpenMP versions supporting SVE. Two Ligra kernels are latency sensitive, whereas SpMV and waLBerla are bandwidth sensitive, making them useful to understand the impact of cross-chiplet latency and system bandwidth. We ran all kernels for 64 threads mapped across 64 cores of each architecture using the default allocation policy. Each memory page is allocated to the local NUMA node of the core, executing the thread upon first touch.

Figure 11 and Figure 12 plot performance comparison and traffic ratio increase due to C2C communication, respectively, where all are normalized to *R1*. Two Ligra kernels show high traffic ratios involving cache coherence protocol exchanges across chiplets: This comes from the nature of graph traversal algorithms relying on the use of compare-and-swap operations. Thus, the performances of both Ligra kernels are highly sensitive to C2C latency, explaining why their performances are lower on *R2* and *R3*, compared to *R1*. In fact, cross-chiplet latencies in *R2* and *R3* are high, as characterized in the core-to-core latency measurements above.

The two bandwidth-hungry kernels, SpMV and waLBerla, show insignificant C2C traffic increases on *R2* and *R3*. This

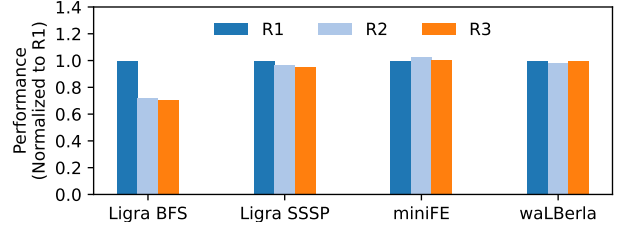


Fig. 11. Performance comparison between *R1*, *R2* and *R3* for Ligra-BFS, Ligra-SSSP, miniFE, and waLBerla. The performances are normalized to the *R1* configuration for each benchmark (larger values indicate better performance).

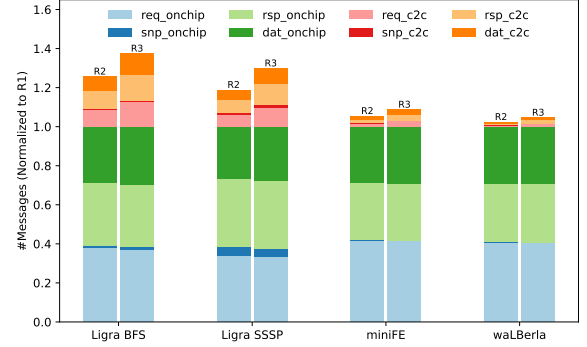


Fig. 12. Breakdown of messages injected into the NoC and those transmitted over the C2C-Link in *R2* and *R3* for various benchmarks. The values on the y-axis are normalized to the *R1* configuration.

is due to less coherence traffic being exchanged at the NoC and C2C interfaces, resulting from limited shared data among threads. Thus, these kernels are less impacted by cross-chiplet latency, and their performances observed on *R2* and *R3* are comparable to those on *R1*.

Co-design insights: (1) Breaking down the monolithic chip (*R1*) into smaller chiplets for a two-chiplet (*R2*) and a four-chiplet (*R3*) design requires at least $4 \times \text{C2CGs}$ and $6 \times \text{C2CGs}$, respectively, along with a sufficient C2C-Link bandwidth of 256 GB/s to improve cross-chiplet bandwidth. (2) In addition, the Transaction Table size within the C2CG structure would need at least 512 entries to mitigate the performance impact of the Retry mechanism employed for routing deadlock avoidance. (3) Lastly, due to the inherent C2C latency, hardware supporting NUMA configurations would benefit for performance optimization on these architectures.

VI. CONCLUSION

The paper presents a reference modeling tool for chiplet-based co-design, especially to help hardware architects in making optimal design choices for Armv8 HPC processors. Our model, built on the gem5 simulator, extends the CHI protocol, and introduces a C2CG architecture to enable chiplet-to-chiplet CHI transfers, cache coherence for data sharing, and deadlock-free chiplet-based architectures. It also supports

⁶The input for both is a Kronecker-generated synthetic graph

C2C-Link configurations with UCle parameters for architectural exploration. We have demonstrated the advantage of the model by exploring three designs for a 64-core chip architecture: monolithic, two-chiplet, and four-chiplet. We believe that the co-design conclusions provide valuable insights for hardware architects, i.e., how to choose optimal C2CGs parameters and sufficient C2C-Link bandwidth, and whether to implement hardware support for NUMA configuration or not.

Our deadlock-free solution using the *Retry* mechanism with in-transit buffers implemented at C2CG has a drawback: the buffer size has to be relatively large holding all outbound packets to mitigate the impact of *RetryAck*. Future work will focus on customizing the C2CG structure to minimize the buffer size while still enabling deadlock-free designs.

Furthermore, we will study the scalability of the C2C model. Therefore we will extend the architecture with a Network-on-Interposer or a network die connecting more than four chiplets. Hence, it is feasible to study other topologies than 2D mesh in the network die and to assemble the chiplets in 3D with advanced packaging technologies.

ACKNOWLEDGMENT

We thank Tiago Muck from Arm Research for his support and valuable discussion.

This work has been performed in the context of the European Processor Initiative (EPI) project, which has received funding from the EuroHPC Joint Undertaking (JU) under Framework Partnership Agreement No 800928 and Specific Grant Agreement No 101036168 (EPI SGA2). The JU receives support from the European Union's Horizon 2020 research and innovation programme and from Croatia, France, Germany, Greece, Italy, Netherlands, Portugal, Spain, Sweden, and Switzerland.

REFERENCES

- [1] G. E. Moore, "Cramming more components onto integrated circuits," *Electronics*, vol. 38, April 1965.
- [2] 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 *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pp. 57–70, June 2021. ISSN: 2575-713X.
- [3] A. Smith, G. H. Loh, M. J. Schulte, M. Ignatowski, S. Naffziger, M. Mantor, M. F. N. Kalyanasundharam, V. Alla, N. Malaya, J. L. Greathouse, E. Chapman, and R. Swaminathan, "Realizing the amd exascale heterogeneous processor vision : Industry product," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, pp. 876–889, 2024.
- [4] H. Sharma, S. K. Mandal, J. R. Doppa, U. Ogras, and P. P. Pande, "Achieving datacenter-scale performance through chiplet-based many-core architectures," in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1–6, 2023.
- [5] M. P. Chan Mok, C. H. Chan, W. C. Shui Chow, Y. Jiao, S. Li, P. Luo, Y. K. Li, and M. Jeong, "Chiplet-based system-on-chip for edge artificial intelligence," in *2021 5th IEEE Electron Devices Technology & Manufacturing Conference (EDTM)*, pp. 1–3, 2021.
- [6] A. W. S. Inc., "Aws graviton technical guide." <https://github.com/aws/aws-graviton-getting-started>, 2024.
- [7] WikiChip, "Aws graviton3 - annapurna labs (amazon)." https://en.wikichip.org/wiki/annapurna_labs/graviton/graviton3, 2021.
- [8] WikiChip, "Aws graviton4 - annapurna labs (amazon)." https://en.wikichip.org/wiki/annapurna_labs/graviton/graviton4, 2023.
- [9] J. Xia, C. Cheng, X. Zhou, Y. Hu, and P. Chun, "Kunpeng 920: The first 7-nm chiplet-based 64-core arm soc for cloud services," *IEEE Micro*, vol. 41, no. 5, pp. 67–75, 2021.
- [10] Thenextplatform, "Intel finally gets chiplet religion with server chips," 2021.
- [11] "Ucie: Universal chiplet interconnect express," 2023. [Accessed 2023-09-27].
- [12] T. Li, J. Hou, J. Yan, R. Liu, H. Yang, and Z. Sun, "Chiplet heterogeneous integration technology—status and challenges," *Electronics*, vol. 9, no. 4, 2020.
- [13] B. Dehlaghi, N. Wary, and T. C. Carusone, "Ultra-short-reach interconnects for die-to-die links: Global bandwidth demands in microcosm," *IEEE Solid-State Circuits Magazine*, vol. 11, no. 2, pp. 42–53, 2019.
- [14] R. Farjadrad and B. Vinnakota, "A bunch of wires (bow) interface for inter-chiplet communication," in *2019 IEEE Symposium on High-Performance Interconnects (HOTI)*, pp. 27–273, 2019.
- [15] C. Consortium, "Ccix base specification rev1.0a v1.0 for evaluation." <https://www.ccixconsortium.com/library/specification/>, 2019. [Accessed 2021-12-13].
- [16] Arm, "Amba chi chip-to-chip (c2c) architecture specification." <https://developer.arm.com/documentation/IHI0098/a>, 2024. [Accessed 2024-12-10].
- [17] Arm, "Arm neoverse cmn-700 coherent mesh network technical reference manual," 2023. [Accessed 2024-07-23].
- [18] R. F. Barrett, S. Borkar, S. S. Dosanjh, S. D. Hammond, M. A. Heroux, X. S. Hu, J. Luitjens, S. G. Parker, J. Shalf, and L. Tang, "On the role of co-design in high performance computing," *Transition of HPC Towards Exascale Computing, Advances in Parallel Computing, IOS Press*, vol. 24, pp. 141–155, 2013.
- [19] M. Sato, Y. Ishikawa, H. Tomita, Y. Kodama, T. Odajima, M. Tsuji, H. Yashiro, M. Aoki, N. Shida, I. Miyoshi, K. Hirai, A. Furuya, A. Asato, K. Morita, and T. Shimizu, "Co-design for a64fx manycore processor and "fugaku"," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–15, 2020.
- [20] P. Majumder, S. Kim, J. Huang, K. H. Yum, and E. J. Kim, "Remote control: A simple deadlock avoidance scheme for modular systems-on-chip," *IEEE Transactions on Computers*, vol. 70, no. 11, pp. 1928–1941, 2021.
- [21] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, p. 1–7, aug 2011.
- [22] J. Smith and J. Doe, *Universal Chiplet Interconnect Express (UCle) Specification Revision 1.1*. Universal Chiplet Interconnect Express, version 1.0 ed., July 2023.
- [23] A. Kannan, N. E. Jerger, and G. H. Loh, "Enabling interposer-based disintegration of multi-core processors," in *2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 546–558, 2015.
- [24] S. Bharadwaj, J. Yin, B. Beckmann, and T. Krishna, "Kite: A family of heterogeneous interposer topologies enabled via accurate interconnect modeling," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2020.
- [25] H. Zheng, K. Wang, and A. Louri, "A versatile and flexible chiplet-based system design for heterogeneous manycore architectures," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2020.
- [26] C. Consortium, "Cache coherent interconnect for accelerators," 2017-2023. [Accessed 2023-09-27].
- [27] E. Taheri, S. Pasricha, and M. Nikdast, "Red: A reliable and deadlock-free routing for 2.5-d chiplet-based interposer networks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 43, no. 12, pp. 4599–4612, 2024.
- [28] J. Yin, Z. Lin, O. Kayiran, M. Poremba, M. Shoaib Bin Altaf, N. Enright Jerger, and G. H. Loh, "Modular routing design for chiplet-based systems," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pp. 726–738, 2018.
- [29] 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 Eight Annual ACM International Conference on Nanoscale Computing and Communication, NANOCOM '21*, (New York, NY, USA), Association for Computing Machinery, 2021.

- [30] 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 ’24, (New York, NY, USA), p. 35–41, Association for Computing Machinery, 2024.
- [31] M. Orenes-Vera, E. Tureci, M. Martonosi, and D. Wentzlaff, “MuchiSim: A Simulation Framework for Design Exploration of Multi-Chip Manycore Systems,” in *2024 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, (Los Alamitos, CA, USA), pp. 48–60, IEEE Computer Society, May 2024.
- [32] T. Krishna and M. Parasar, “gem5 chips,” https://github.com/GT-CHIPS/gem5_chips, 2019. [Accessed 2024-12-13].
- [33] Arm, “Arm neoverse v1 reference design software developer guide,” 2021. [Accessed 2023-10-27].
- [34] J. Defilippi, “Towards chiplet interoperability with amba chi c2c,” Open Compute Project Chiplet Summit, 10 2023.
- [35] L. Zaourar, M. Benazouz, A. Mouhagir, F. Jebali, T. Sassolas, J.-C. Weill, C. Falquez, N. Ho, D. Pleiter, A. Portero, E. Suarez, P. Petrakis, V. Papaefstathiou, M. Marazakis, M. Radulovic, F. Martinez, A. Arme-jach, M. Casas, A. Nocua, and R. Dolbeau, “Multilevel simulation-based co-design of next generation hpc microprocessors,” in *2021 International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*, pp. 18–29, 2021.
- [36] J. Flich, P. Lopez, M. Malumbres, J. Duato, and T. Rokicki, “Applying in-transit buffers to boost the performance of networks with source routing,” *IEEE Transactions on Computers*, vol. 52, no. 9, pp. 1134–1153, 2003.
- [37] C. Falquez, “gem5-dbc,” <https://github.com/FZJ-JSC/gem5-dbc>, 2024.
- [38] P. John D. McCalpin, “Stream: Sustainable memory bandwidth in high performance computers,” [Accessed 2023-10-27].
- [39] M. A. Heroux, D. W. Doerfler, P. S. Crozier, J. M. Willenbring, H. C. Edwards, A. Williams, M. Rajan, E. R. Keiter, H. K. Thornquist, and R. W. Numrich, “Improving Performance via Mini-applications,” Tech. Rep. SAND2009-5574, Sandia National Laboratories, 2009.
- [40] C. Feichtinger, J. Götz, S. Donath, K. Iglberger, and U. Rüde, *WalBerla: Exploiting Massively Parallel Systems for Lattice Boltzmann Simulations*, pp. 241–260. Springer, London, 06 2009.
- [41] J. Shun and G. E. Blelloch, “Ligra: a lightweight graph processing framework for shared memory,” in *Proceedings of the 18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPOPP ’13, (New York, NY, USA), p. 135–146, Association for Computing Machinery, 2013.
- [42] C. Falquez, “Instruction execution throughput microbenchmarks,” <https://github.com/FZJ-JSC/ietubench>. commit d84ca4a.
- [43] N. Viennot, “Measuring cpu core-to-core latency,” <https://github.com/nviennot/core-to-core-latency?tab=readme-ov-file>, 2024. [Accessed 2024-12-13].

VII. APPENDIX

Assumption 1. Any chiplets used in a C2C interconnect are assumed to be deadlock-free networks. In addition, transmissions via C2C-Link are assumed to be non-blocking.

Rule 1. Inside a chiplet, it should be possible to temporarily eject any outbound packet at a gateway (C2CG), once injected into the network. This means that when an outbound packet reaches the C2CG, the C2CG will check for the availability of at least one buffer capable of storing the entire packet. If a buffer is available, the packet is queued. Otherwise, the packet will be discarded, and the C2CG must send a *RetryAck* packet to the source router for the outbound packet to be resent in the future.

Theorem 1. The C2C interconnection is deadlock-free if outbound packet transfers adhere to Rule 1.

Proof. Assume a C2C interconnection has N packet buffers at a C2CG, and there are M C2CGs. If a deadlock occurs,

there must be a cycle involving $n \geq 1$ outbound packets still inside the chiplet. Other packets in the cycle are outbound packets already outside the chiplet, and are either inbound or intra packets. According to **Rule 1**, n must satisfy $n \leq N \times M$. Since the intra-chiplet network is deadlock-free, and it is possible to eject the n outbound packets at the C2CGs, the cycle can not be blocked. This contradicts the assumption of deadlock. \square