Server Chiplet Networking

Seunghyun An
University of Wisconsin-Madison
Madison, WI, USA

Joontaek Oh
University of Wisconsin-Madison
Madison, WI, USA

Ming Liu
University of Wisconsin-Madison
Madison, WI, USA

Abstract

Emerging chiplet-based server platforms and the resulting server chiplet networking present a fundamental shift in the (intra-)host network. Unlike conventional monolithic servers, compute chiplets, I/O chiplets, off-chip memory, and peripheral devices communicate through a collection of heterogeneous interconnects and links, formalizing a new server chiplet network substrate that has not been explored before. This paper makes an initial step by characterizing two generations of AMD EPYC chiplet servers, identifying four communication idiosyncrasies, and summarizing the design implications. We outline some future directions under server chiplet networking and discuss how to build next-generation server systems and applications.

CCS Concepts

• **Computer systems organization → Interconnection architectures**; • **Hardware → Network on chip**; • **Networks → Network performance analysis**.

Keywords

Chiplet, Network on chip, Performance analysis, Host network

ACM Reference Format:
Seunghyun An, Joontaek Oh, and Ming Liu. 2025. Server Chiplet Networking. In *The 24th ACM Workshop on Hot Topics in Networks (HotNets '25), November 17–18, 2025, College Park, MD, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3772356.3772389

1 Introduction

Chiplet [33, 37, 64, 89, 90, 96] is an emerging semiconductor technology and has garnered significant attention in industry and academia. By dividing monolithic large chips into domain-specific, small, modular dies and composing them via silicon interposers, chiplets drastically improve the wafer yield rate, reduce manufacturing costs, expedite time-to-market, facilitate flexible and scalable hardware innovations, and enable energy- and cost-efficient application specialization. The past few years have seen a number of chiplet-based processors [10, 11, 17, 57, 61, 85, 86], domain-specific hardware accelerators [15, 28, 87], and I/O devices [34, 68, 69].

Commodity servers are increasingly built using chiplets. A server SoC (system-on-chip) consists of several compute and I/O chiplets, connected via specialized on-chip load-store interconnects. A compute chiplet encompasses a few to dozens of cores, sometimes organized as sub-chiplets, sharing per-chiplet last-level cache slices. An I/O chiplet, enclosing a network-on-chip (NoC), provides high-bandwidth and low-latency network connectivity for the memory subsystem and I/O devices. As such, a new type of host network–where we term it **Server Chiplet Networking**–emerges, which serves as the communication subsystem underlying the server SoC.

Server chiplet networking is essentially a network of heterogeneous networks, usually organized into three layers. The physical layer encompasses several different links, such as on-chip cache-coherent interconnects (like AMD Infinity Fabric [3] and UCIe [18]), off-chip memory interconnects, and peripheral I/O buses. The data link layer is a reliable and hierarchical packet-switched network whose topology hinges on the NoC of an I/O chiplet. The transaction layer provides communication semantics and deterministically routes data FLITs from the source to the destination. *Server chiplet networking entails communication characteristics that are drastically different from those of a traditional monolithic SoC, an area that has not been previously explored.*

This paper makes the first step by systematically characterizing two generations of AMD EPYC chiplet servers (§3). We identify four unique aspects of server chiplet networking, i.e., extended data paths with more latency hops, heterogeneous bandwidth domains, inconsistent bandwidth-delay products, and sender-driven aggressive bandwidth partitioning, and discuss the design implications. We believe a networking stack and corresponding ecosystem centered around chiplet networking are strongly needed, and will offer several benefits, including improved communication, greater efficiency, maximized computing usage, and enhanced development capabilities. We conclude the paper by outlining future research directions with some proposals (§4).

HotNets '25, College Park, MD, USA

ACM ISBN 979-8-4007-2280-6/25/11
https://doi.org/10.1145/3772356.3772389

2 Background and Motivation

This section provides some necessary background on chiplets, describes the chiplet-based server SoC (system-on-chip) architecture, and introduces server chiplet networking.

2.1 Chiplet and Why

Chiplet [33, 37, 64, 89, 90, 96], an emerging semiconductor technology, has become a predominant approach to building chips. A chiplet is a discrete and unpackaged modular die that implements specific functionalities, such as processors, memory controllers, I/O subsystems, and acceleration kernels. It provides standardized operating interfaces and is integrated into a compatible 2.5D/3D silicon interposer [99, 100]. The interposer, acting as a bridge, facilitates low-latency and high-speed die-to-die data transfers via specialized chiplet interconnects, such as UCIe [18], BoW [74], and OpenHBI [73]. Today, we have seen several chiplet-based computing products designed, manufactured, and deployed, such as AMD EPYC [10, 11, 57, 58, 85, 86], Intel Xeon Scalable Processor [56, 59, 91], AWS Graviton [61], Ampere AmpereOne [17], NVIDIA H100/H200 [15, 28], and AMD MI300 [87].

Chiplets impose several benefits. First, they improve the yield rate by breaking a large chip into smaller dies, as the defect probability increases with the die size. Considering a standard $360mm^2$ sized wafer and a typical die-yielding model [58, 98], a 4-chiplet ($99mm^2$) design can achieve a yield rate of 37%, doubling that of a monolithic approach (15%). Such improvements become more significant under small process nodes (like 3/5nm). Second, chiplets can be fabricated with different process nodes and foundries, and reused across different hardware substrates, drastically minimizing the time-to-market and reducing the manufacturing cost. Third, they make building flexible and scalable heterogeneous SoCs possible, enabling fast innovation and agility based on the application demands. For example, the recent Intel SDV (software-defined vehicle) SoC [34] employs a multi-node chiplet solution that allows automakers to tailor compute, graphics, and AI capabilities flexibly. Fourth, designers can explore application specialization, apply a Lego-style strategy to design complex SoCs, and freely upgrade/add/drop new functionalities, maximizing the power density and energy efficiency. The Cadence Allegro X platform [12] leverages this for advanced IC packaging design.

2.2 Chiplet-based Server SoC

Most recent server processors are developed using chiplets. Take the AMD EPYC 7302 processor as an example (others are similar). As shown in Figure 1, the server SoC has (a) four compute chiplets or Core Complex Die (CCD) in the AMD terminology and (b) one I/O chiplet. A compute chiplet contains several sub-chiplets (2 in our example) or

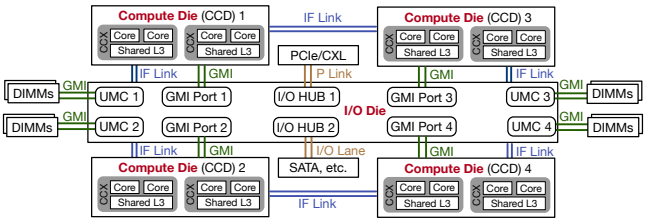


Figure 1: An architecture overview of a chiplet-based server SoC. We take the AMD EPYC processor as an example. CCD = Core Complex Die. GMI=Global Memory Interconnect. IF=Infinity Fabric. UMC=Unified Memory Controller.

Parameters	EPYC 7302	EPYC 9634
Microarchitecture	Zen 2	Zen 4
L1 (per core)	32KB	64KB
L2 (per core)	512KB	1MB
L3 (per CPU)	128MB	384MB
Core#/CCX#/CCD# (per CPU)	16/8/4	84/12/12
Compute Chiplets # (per CPU)	4	12
Process technology (Compute Die)	7nm	5nm
I/O Chiplets # (per CPU)	1	1
Process technology (I/O Die)	12nm	6nm
PCIe Gen/Lane #	Gen4/128	Gen5/128
Base/Turbo Frequency	3/3.3 GHz	2.25/3.7 GHz

Table 1: HW specifications of our two evaluated processors.

Core Complex (CCX) that share the last-level cache slices. A CCX consists of one to dozens of cores, each of which has its own L1 and L2 caches. An I/O chiplet encompasses (a) UMCs (unified memory controllers) that connect to off-chip DIMMs; (b) I/O hubs that provide peripheral links for devices, such as P Links to fast PCIe/CXL slots and I/O lanes to slow SATA-like buses; (c) GMI (global memory interconnect) ports for compute chiplets to access memory; and (d) a mesh network that interconnects different components through some proprietary routing protocols. Compute-Compute and Compute-I/O chiplets talk to each other via the AMD Infinity Fabric or other specialized interconnects like UCIe [18]. Off-chip DIMMs also attach to UMCs through GMI links.

Table 1 presents two AMD EPYC chiplet processors we studied in this paper. Note that the 7302 CPU contains two CCXs per CCD, while the 9634 processor only has one.

2.3 Server Chiplet Networking

Server chiplet networking is the communication subsystem underlying the server SoC that connects different microarchitectural modules and devices. Its physical (*L1*) layer is an agglomeration of on-chip interconnects, off-chip memory links, and peripheral I/O buses, providing connectivity for data flows traversing compute chiplets, I/O chiplets, and devices. Next, the link (*L2*) layer is a reliable and hierarchical packet-switched network. As shown in Figure 2, the first level is the network-on-chip (NoC) in an I/O chiplet, employing a Mesh [31], Torus [55], Cube [25], or Dragonfly [38]

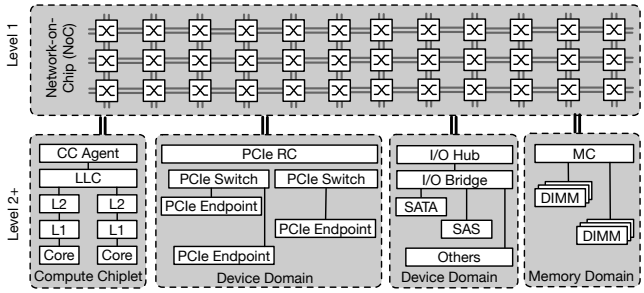


Figure 2: A topological view of server chiplet networking.

topology. The following levels are organized in different compute chiplets or device domains, usually presenting a tree topology. The network contains different switches or routers that use either bufferless or buffered routing protocols. Last, the transaction (*L3*) layer describes data flows from source to destination entities at the cacheline or FLIT granularity, depending on the underlying link. For example, a cacheline-sized LLC snooping request mostly traverses the Infinity Fabric, while a `CXL.mem` transaction, encoded as the FLIT size (68/256B), goes from a compute chiplet and I/O chiplet to a CXL DIMM through an IF, a P Link, and a CXL lane.

Server chiplet networking has become crucial given (a) the proliferation of dense and heterogeneous hardware-accelerated computing boxes [23, 24, 62, 63], (b) increasing interconnect speeds for fast cross-device (domain) data movements [3, 18], and (c) skyrocketing application demands [50]. A new networking stack and corresponding ecosystem (including libraries, utilities, and runtime) are strongly needed, offering several benefits. First, it provides the opportunity to maximize the communication performance under today's sub-microsecond and terabit regime, which is notoriously challenging [4, 40]. Second, it can improve the overall system energy efficiency by accelerating data transfer and streamlining computation-communication overlapping close to the underlying hardware tiers. Third, it facilitates the system's compute, memory, and I/O scalability, avoids superfluous scaling bottlenecks (due to contention and head-of-line blocking), and ameliorates the multi-tenancy support under massive intra-server data flows. Fourth, it enables microscopic per-request and per-flow observability for system diagnostics, anomaly detection, performance profiling, and upper-layer development. **However, our community lacks such a system layer and the capabilities it would provide.**

3 Understanding Server Chiplet Networking

We systematically characterize server chiplet networking, report our findings, and discuss the design implications.

		EPYC 7302	EPYC 9634
Compute Chiplet	L1	1.24 ns	1.19 ns
	L2	5.66 ns	7.51 ns
	L3	34.3 ns	40.8 ns
	Max CCX Q	30 ns	20 ns
	Max CCD Q	20 ns	N/A
I/O Chiplet	Switching Hop	~8 ns	~4 ns
	I/O Hub	~15 ns	~15 ns
Memory/Device	Near	124 ns	141 ns
	Vertical	131 ns	145 ns
	Horizontal	141 ns	150 ns
	Diagonal	145 ns	149 ns
	CXL DIMM	N/A	243 ns

Table 2: The data path latency breakdown. We measured the latency by configuring the pointer-chasing mode of our utility and gradually increasing the working set. We changed the NPS (Node per Socket) configurations and issued memory requests to DIMMs at different positions.

3.1 Experimental Methodology

We use two types of commodity servers for the characterization experiments. The Dell 7525 2U box contains two EPYC 7302 processors and 256GB DDR4. The Supermicro 1U server has one EPYC 9634 processor, 1TB DDR5, and four Micron CZ120 CXL modules (256GB each). Both servers run Ubuntu 22.04. We developed a micro benchmark utility (like others [6]) that can flexibly generate different data flows (such as one or multiple concurrent cachelines, random/sequential read/write access patterns, and temporal or non-temporal writes) over a size-configurable working set, originating from and destined to compute chiplets, memory domains, and device domains across the chiplet networking subsystem. We mostly use latency and bandwidth as performance metrics.

3.2 Extended Data Path with More Latency Hops

Server chiplets add more intermediate hops compared with monolithic ones, increasing the data communication path and incurring some latency overhead. Within a compute (sub)-chiplet, there is a traffic control module that limits the number of outstanding requests. It employs a queueless structure (like Phantom Queue [1]) and uses tokens and back-pressure for overload control. We observe up to 50ns and 20ns queueing delay in an EPYC 7302 and EPYC 9634 processor (Table 2). Inside an I/O chiplet, requests traverse through a sequence of micro-architectural modules before reaching the target DIMM/device (Figure 2), including a cache-coherent master (CCM), several I/O chiplet switching hops (SHops), an I/O hub, a coherent station (CS), and a unified memory controller (UMC). The switching hop and I/O hub take roughly 8ns and 15ns on the EPYC 7302 (4ns and 15ns in the EPYC 9634). Thus, accessing a DIMM at a near, vertical, horizontal, and diagonal position (relative to the compute chiplet) yields different latencies, i.e., 124ns (141ns), 131ns (145ns), 141ns (150ns), and 145ns (149ns) on an EPYC 7302 (9634), because

	To DIMM Read/Write (GB/s)		To CXL Read/Write (GB/s)	
	EPYC 7302	EPYC 9634	EPYC 7302	EPYC 9634
From Core	14.9/3.6	14.6/3.3	N/A	5.4/2.8
From CCX	25.1/7.1	35.2/23.8		23.6/15.8
From CCD	32.5/14.3			
From CPU	106.7/55.1	366.2/270.6		88.1/87.7

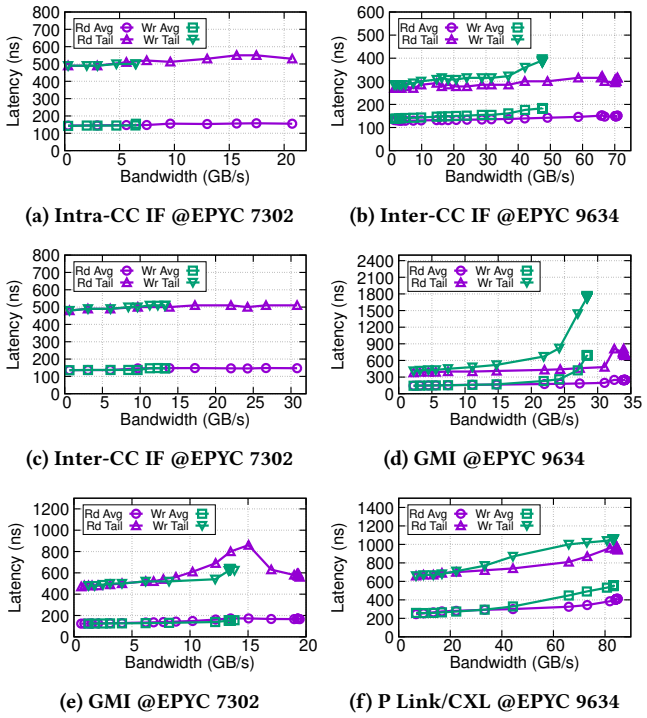
Table 3: Maximum achieved bandwidth from a core/CCX/CCD/CPU when accessing the DIMMs and CXL device. We measured the bandwidth using AVX512 memory read and non-temporal write operations.

the number of traversed hops varies. Accessing a PCIe device not only goes through the I/O hub, but also passes through a PCIe root complex, an I/O moderator (like an I/O northbridge), and P Links. Hence, a cacheline-sized CXL memory access on the EPYC 9634 takes 243ns.

Implication #1: Chiplet servers generally extend the data access path among cores, DIMMs, and devices because of intra/inter-chiplet routing, imposing higher memory and I/O stall cycles. It will affect traditional memory-sensitive primitives, such as memory fences, synchronization locks, and MMIOs. We would also see more granular non-uniform memory access, such as the Sub-NUMA Clustering feature. Asynchronous threading/coroutines, wait-free/lock-free data structures, and locality-aware data placement are becoming much more favorable to elude superfluous memory stalls and maximize core utilization. Recent work on memory-bound stall cycle harvesting [49] is also promising.

3.3 Heterogeneous Bandwidth Domains

Server chiplet networking is an agglomeration of heterogeneous networks that feature different bandwidth domains. The achieved bandwidth is determined by the number of concurrent transactions issued end-to-end over the underlying links, including Infinity Fabric (IF), Global Memory Interconnect (GMI), I/O chiplet internal interconnect, P link, and PCIe/CXL. As shown in Table 3, a core sustains a read/write bandwidth of 14.9/3.6 GB/s and 14.6/3.3 GB/s to DIMMs on an EPYC 7302 and 9634 CPU, respectively, limited by the per-core memory-level parallelism. When using all cores on a compute chiplet, memory read/write bandwidths reach up to 32.5/14.3 GB/s and 35.2/23.8 GB/s, respectively, constrained by the compute chiplet's GMI link capacity. Given that a unified memory controller (UMC) can deliver at most 21.1/19.0 GB/s and 34.9/28.3 GB/s of read/write bandwidth, a compute chiplet must access multiple memory controllers to attain higher aggregated bandwidth. We then activate all 4 and 12 compute chiplets in the two CPUs and issue as many memory accesses as possible to all DIMMs. An EPYC 7302 achieves a peak read/write throughput of 106.7/55.1 GB/s, and an EPYC 9634 attains up to 366.2/270.6 GB/s, both limited by the NoC routing capacity in the I/O chiplet.



(a) Intra-CC IF @EPYC 7302 (b) Inter-CC IF @EPYC 9634

(c) Inter-CC IF @EPYC 7302 (d) GMI @EPYC 9634

(e) GMI @EPYC 7302 (f) P Link/CXL @EPYC 9634

Figure 3: We report the average and tail (P999) latency when varying the bandwidth of the Infinity Fabric (IF), Global Memory Interconnect (GMI), and P Link/CXL on the EPYC 7302 and 9634 processors. We vary the traffic load by issuing sequential read and non-temporal write operations and use `NOP` instructions to control the rate. CC=Compute Chiplet.

Further, when accessing a peripheral device, the I/O path segment (like the I/O hub, P link, and PCIe lanes) can become a bottleneck. For example, on an EPYC 9634, a single core, a single compute chiplet, and all 12 chiplets reach 5.4/2.8 GB/s, 23.6/15.8 GB/s, and 88.1/87.7 GB/s of CXL memory read/write bandwidth, respectively, which is 63.0%/22.2%, 33.0%/33.6%, and 78.1%/69.3% lower than the local DIMM performance due to the high access latency and PCIe/CXL bandwidth limit.

Implication #2: Memory wall [65, 66] is a well-known problem that denotes the bandwidth gap between computing engines and memory subsystems. In chiplet server systems, an emerging hidden "interconnect wall" would happen either within or across chiplets, limiting the data movement speed even before saturating the memory bandwidth. The issue would become common in high-bandwidth servers with terabit I/Os. Thus, being aware of end-to-end bandwidth domains, identifying the bandwidth throttling path segment at runtime, and developing an intra-server traffic matrix [51, 92] are essential for maximizing the data transmission performance. Besides, we expect that more new computational devices that support near-data computing [79, 82, 94] and

distributed and collaborative computing [7, 22, 46] will be designed and implemented.

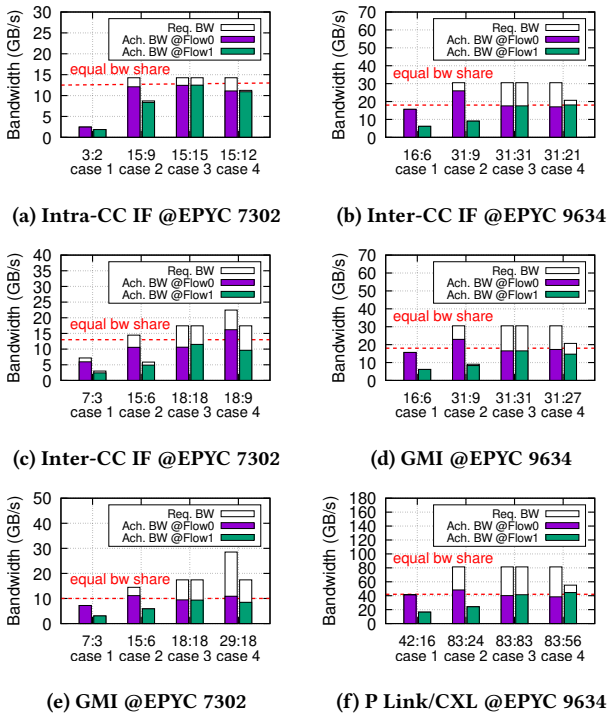
3.4 Inconsistent Bandwidth-Delay Product (BDP)

Memory and I/O requests in chiplet servers traverse several different interconnects with incongruent BDPs. When the number of in-flight transactions exceeds the underlying link bandwidth capacity, one sees queuing or back-pressure overheads, yielding head-of-line blocking. Figure 3 presents our characterizations of different interconnects. Regarding Infinity Fabric, the EPYC 7302 CPU provisions enough bandwidth for intra- and inter-compute chiplet scenarios (Figures-3-a/c). Thus, the average/tail read and write latencies are 144.5ns/490.0ns and 142.5ns/500.0ns, regardless of the load. However, the EPYC 9634 is less-provisioned, where one compute chiplet contains 7 cores, and we see a 2× latency increase when approaching the max bandwidth (Figure 3-b). In terms of GMI, since it connects on-chip memory controllers to off-chip DIMMs through an I/O chiplet, the per-GMI channel bandwidth is less than the on-chip interconnect, causing significant request buffering. As shown in Figures 3-d/e, the memory read average/tail latencies of an EPYC 7302 (9634) rise to 172.5ns/800.0ns (249.5ns/810.2ns) from 123.7ns/470.0ns (143.7ns/380.0ns), respectively. And the write ones increase from 123.9ns/480.0ns (144.1ns/350.2ns) to 153.5ns/630.0ns (695.8ns/1749.8ns). The problem also happens with the P Link/CXL. Since the I/O chiplet on an EPYC 9634 is capable of driving 366.2/270.6 GB/s read/write traffic (Table 3), we observe 1.7/1.4× and 2.1/1.6× average/tail read and write latency increases (Figure 3-f), respectively.

Implication #3: Compared with conventional monolithic servers, chiplet servers generally embody a much larger BDP for data transmission among CPUs, DIMMs, and devices, and can handle more outstanding transactions. However, inconsistent BDP of different links would then incur queuing and head-of-line blocking, jeopardizing bandwidth usage and wasting computing cycles. Dynamic monitoring end-to-end runtime BDP and using it for traffic control becomes vital in server chiplet networking. Akin to rate limiters and traffic policers in today's OS networking and I/O stacks [32, 39, 77, 80], we expect such designs will also be developed and applied to inter-/intra- chiplet communications.

3.5 Sender-driven Aggressive Bandwidth Partitioning

Next, we examine how bandwidth is partitioned in server chiplet networking. We launch two competing flows at different links, use `NOP` instructions to control their requested bandwidth, and see how much bandwidth each flow achieves. When the link is under-subscribed, as expected, both flows



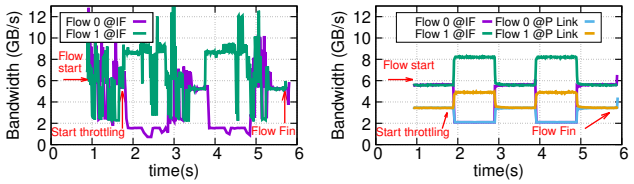
(a) Intra-CC IF @EPYC 7302 (b) Inter-CC IF @EPYC 9634

(c) Inter-CC IF @EPYC 7302 (d) GMI @EPYC 9634

(e) GMI @EPYC 7302 (f) P Link/CXL @EPYC 9634

Figure 4: Bandwidth (BW) partitioning of two competing data flows at a shared link on the EPYC 7302 and 9634 processors. We examine four cases. The first targets when the aggregated requested bandwidth is less than the link capacity, while the other three consider the over-subscription scenario. Case 2 has one flow whose requested bandwidth is less than the equal share. In cases 3 and 4, two flows request the same and different amounts of bandwidth (more than their equal share), respectively. CC=Compute Chiplet.

can receive the requested bandwidth, regardless of the link type (case 1 in Figure 4). When the link is over-subscribed, i.e., the aggregated requested bandwidth exceeds the bandwidth capacity, we find that the bandwidth partition follows a sender-driven aggressive manner (cases 2 and 4). In particular, the flow with a higher demand takes more bandwidth than its equal share. This is because each communication intermediate point at a compute or I/O chiplet is unaware of (a) what a flow is and (b) what the demand of a flow is. As a result, such traffic-oblivious routing always benefits an aggressive sender that pushes more requests in-flight. Two flows with the same demand receive the equilibrium bandwidth (case 3). Further, we evaluate the link bandwidth usage when its housed flows have unsteady demands. Specifically, as shown in Figure 5, we reduce the traffic rate of flow 0 by 2.0GB/s at the 2–3s and 4–5s periods, and observe that flow 1 can reliably take the unused bandwidth at the IF and P Link on an EPYC 9634 processor. However, the EPYC 7302 sees drastic variation at the IF link (we suspect this is due to the



(a) EPYC 7302 (b) EPYC 9634

Figure 5: Bandwidth utilization of two competing flows with fluctuating demands. We throttle flow 0 and see whether flow 1 (unthrottled) can reap the unused bandwidth.

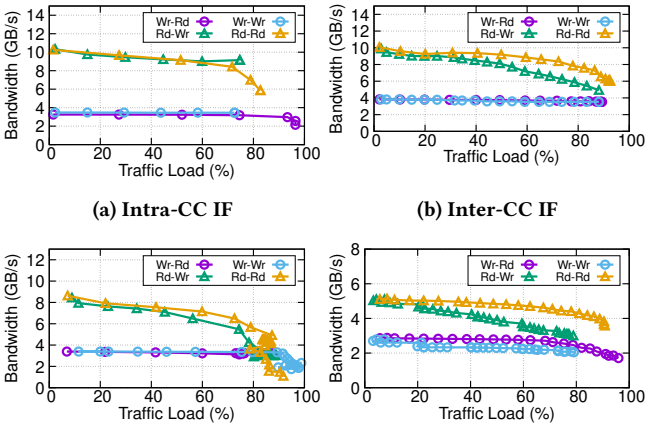
intra-CC queuing module). Bandwidth harvesting does not happen instantly. It takes roughly 100ms and 500ms for the EPYC 9634 to reap unused bandwidth on the IF and P Link, respectively. When flow 0 finishes throttling, the two flows would again take an equal bandwidth share.

When read/write data streams mix, we observe that interference occurs only when a particular link in one direction is saturated. As shown in Figure 6, within a compute chiplet over IF, writes and reads are affected when the background read stream approaches 32.8GB/s and 27.7GB/s. The background write stream induces little interference. Across the compute chiplets, we observe that data stream performance is affected at much higher bandwidth, because the I/O chiplet provisions more than one routing path. For example, the write flow is rarely affected regardless of the background traffic, while reads are degraded when the aggregated bandwidth exceeds 55.7GB/s. At the GMI and P Link/CXL, interference occurs when the aggregated read(write) bandwidth reaches 31.8(29.1) GB/s and 62.8(44.0) GB/s.

Implication #4: Traffic-oblivious bandwidth allocation and routing are the de facto traffic control scheme. This works well for traditional monolithic servers whose NoC bandwidth is over-provisioned. However, this is no longer the case for server chiplet networking, where compute and I/O chiplets might contend for one or several links. Thus, it will be valuable to introduce the communication flow abstraction, materialize it in a global software-based traffic manager, and expose it to the chiplet network. In this way, one could develop application-specialized traffic control instead of relying on the sender side naively. Read and write interference would also occur when an interconnect link in one direction is over-subscribed. Recent studies on host networking [95] offer a promising approach to build upon.

4 Looking Forward

Server chiplet networking changes how processors, memory, and I/O devices communicate across a server platform, entailing unique performance characteristics. We outline new research directions on grappling with its ramifications.



(a) Intra-CC IF (b) Inter-CC IF

(c) GMI (d) P Link/CXL

Figure 6: Read/write interference at the IF, GMI, and P Link/CXL on the EPYC 9634 processor. We run a frontend stream X at max rate, vary the traffic load of the background one Y, and report how much bandwidth X achieves (X-Y).

#1: Hardware-abstracted Chiplet Networking Layer. `Device tree` [21] is a data structure that describes the hardware components and their organizational structure of an embedded/PC/server platform, exposed to the operating system for system management and maintenance. We believe that a similar hardware abstraction for chiplet networks (like `/sys/firmware/chiplet-net`) is essential. It not only presents an architectural overview (as Figure 1), but also provides runtime performance telemetry statistics for each link and intermediate hop through `/proc/chiplet-net`, facilitating low-level system development.

#2: Rethinking Scalable Operating System Design. With hardware modularity, chiplet servers drastically increase the number of cores (e.g., a four-socket AmpereOne server featuring 1024 cores), motivating us to revisit the design principles of many-core operating systems [5, 8, 9, 16]. For example, the multikernel [5] OS structure is motivated by the costly interconnect (i.e., AMD HyperTransport) when handling cache coherence on a shared memory system. It makes inter-core communication explicit and uses asynchronous messages to synchronize replicated states. However, such a design might not be suitable in chiplet networking due to the extended communication path (§3.2), heterogeneous bandwidth domains (§3.3), and inconsistent BDP (§3.4). It is pivotal to explore, refine, or perhaps redefine what the scalable commutative rule [16] is.

#3: Fused Intra-/Inter-Host Networking and I/O Stack. Recent trends indicate that inter-fabric bandwidth has gradually approached or even outpaced intra-host bandwidth, like in disaggregated storage [27, 32, 35, 36, 45, 52–54, 105]. For example, a 400+GbE terabit Ethernet port and 8+ NVMe SSDs

can sometimes drive more bandwidth than a compute chiplet. The problem becomes even worse if considering the inconsistent BDP (§3.4) and unregulated bandwidth partitioning (§3.5). Researchers have partly tackled this issue [13, 32]. For example, NetChannel [13] introduces a disaggregated networking stack and uses a flexible channel abstraction for streaming data in and out. `blk-switch` [32] proposes a switching architecture for the block layer for efficient multi-tenancy. In chiplet networking, the network and I/O stack should consider both the internal and external link characteristics and judiciously orchestrate data flows across compute chiplets, I/O chiplets, memory domains, and devices.

#4: Intra-host Switching for Accelerators. Dense GPU and domain-specific accelerator servers have become prevalent in the era of AI. Although massive data communications use dedicated external fabrics (like NVLink), server chiplet networking is an important component for the signal plane and host-accelerator interaction. Specifically, the accelerator execution is activated via submission commands and completed through acknowledgment responses, which are latency-sensitive. Bandwidth-intensive input/output data is copied to/from the accelerator memory explicitly through DMA or data movement engines, managed by the host CPU. In chiplet networking, all such communications traverse the device bus, I/O hub, and I/O chiplet, which embody performance idiosyncrasies (§3.2–§3.5). To fully utilize the accelerator and eliminate movement-induced stalls, one should develop an intra-host switching module that proactively monitors the traffic matrix [51], conceives an optimal communication path and schedule, and provisions just enough bandwidth. Researchers have explored such a design in programmable networks [7, 14, 22, 42, 75, 76, 83, 84, 104, 106] and in-network computing [29, 44, 46–48, 67, 81].

#5: Chiplet-Centric System Modeling, Benchmarking, and Profiling. Chiplet networking significantly complicates server operational observability, making resource provisioning, performance diagnostics, and debugging challenging. Besides the sub-microsecond granularity, difficulties arise from much more intertwined micro-architectural components, with very limited hardware monitoring counters. We propose to take an interconnect transaction view and develop a chiplet-centric architectural performance model, as many others [2, 19, 26, 43, 101, 102], to capture both data and computing flows. Next, it would be useful to develop a benchmarking framework [30, 60] for cross-platform systematic characterization and to produce practical guidelines. Last, we advocate for a system-level `perf`-like [20, 41] profiling utility, entrenched with the server SoC, that collaboratively combines the hardware architectural PMU (Performance Monitoring Unit) with time-series-based probabilistic and compact data structures (like Sketches [88, 97, 103]) to distill application-specific execution telemetry.

#6: From Server Chiplet Networking to Accelerator Chiplet Networking. As accelerators are increasingly built using chiplets, we believe that accelerator chiplet networking will also be an interesting exploration area. In addition to the idiosyncrasies uncovered in §3, taking existing AI accelerators [70–72, 78, 93, 107] as an example, we expect that (a) the tighter coupling between communication and computation (such as tensor tiles, activation gradient, and attention maps), (b) mixed dataflows (like control-plane kernel dispatching and data-plane tensor streaming), and (c) multi-tier communication hierarchy (e.g., inter-chiplet, on-package NOC, and inter-packet) will impose more challenging issues, requiring us to rethink traffic control, kernel scheduling, and communication collective.

5 Conclusion

This paper introduces a new type of host network–server chiplet networking. We perform a systematic characterization of AMD chiplet-based EPYC server platforms and summarize several key design implications. We outline future research directions, and believe that server chiplet networking will fundamentally shape the design and implementation of future performant, efficient, and scalable server systems.

Acknowledgement

We would like to thank the anonymous reviewers and our shepherd, Jana Iyengar, for their comments and feedback. This work is supported in part by NSF grants CNS-2106199, CNS-2212192, and CAREER-2339755.

References

[1] Mohammad Alizadeh, Abdul Kabbani, Tom Edsall, Balaji Prabhakar, Amin Vahdat, and Masato Yasuda. 2012. Less is more: trading a little bandwidth for ultra-low latency in the data center. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation (NSDI'12)*.

[2] Muhammad Shoaib Bin Altaf and David A. Wood. 2017. LogCA: A High-Level Performance Model for Hardware Accelerators. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*.

[3] AMD. 2025. AMD Infinity Fabric. https://www.amd.com/content/dam/amd/en/documents/instinct-tech-docs/other/56978.pdf.

[4] Luiz Barroso, Mike Marty, David Patterson, and Parthasarathy Ranganathan. 2017. Attack of the killer microseconds. *Commun. ACM* 60, 4 (2017), 48–54.

[5] Andrew Baumann, Paul Barham, Pierre-Evariste Dagand, Tim Harris, Rebecca Isaacs, Simon Peter, Timothy Roscoe, Adrian Schüpbach, and Akhilesh Singhania. 2009. The multikernel: a new OS architecture for scalable multicore systems. In *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles (SOSP'09)*. 29–44.

[6] Timo Bingmann. 2013. PMBW. http://panthema.net/2013/pmbw/.

[7] Pat Bosshart, Glen Gibb, Hun-Seok Kim, George Varghese, Nick McKeown, Martin Izzard, Fernando Mujica, and Mark Horowitz. 2013. Forwarding metamorphosis: fast programmable match-action processing in hardware for SDN. In *Proceedings of the ACM SIGCOMM*

2013 Conference on SIGCOMM (SIGCOMM'13). 99–110.

[8] Silas Boyd-Wickizer, Haibo Chen, Rong Chen, Yandong Mao, Frans Kaashoek, Robert Morris, Aleksey Pesterev, Lex Stein, Ming Wu, Yuehua Dai, Yang Zhang, and Zheng Zhang. 2008. Corey: an operating system for many cores. In *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation (OSDI'08)*. 43–57.

[9] Silas Boyd-Wickizer, Austin T. Clements, Yandong Mao, Aleksey Pesterev, M. Frans Kaashoek, Robert Morris, and Nickolai Zeldovich. 2010. An analysis of Linux scalability to many cores. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation (OSDI'10)*. 1–16.

[10] Thomas Burd, Wilson Li, James Pistole, Srividhya Venkataraman, Michael McCabe, Timothy Johnson, James Vinh, Thomas Yiu, Mark Wasio, Hon-Hin Wong, Daryl Lieu, Jonathan White, Benjamin Munger, Joshua Lindner, Javin Olson, Steven Bakke, Jeshuah Sniderman, Carson Henrion, Russell Schreiber, Eric Busta, Brett Johnson, Tim Jackson, Aron Miller, Ryan Miller, Matthew Pickett, Aaron Horiuchi, Josef Dvorak, Sabeesh Balagangadharan, Sajeesh Ammikkallingal, and Pankaj Kumar. 2022. Zen3: The AMD 2nd-Generation 7nm x86-64 Microprocessor Core. In *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, Vol. 65. 1–3.

[11] Thomas Burd, Srividhya Venkataraman, Wilson Li, Timothy Johnson, Jerry Lee, Srikirti Velaga, Mark Wasio, Thomas Yiu, Franklin Bodine, Michael McCabe, Udin Salim, Santosh Kumar Thouta, Michael Golden, Sowmya Ramachandran, Gokul Subramani Lakshmi Devi, John Wuu, Yarek Kuszczak, Gaurav Singla, Carson Henrion, Andy Robison, Sabeesh Balagangadharan, Umesh Nair, Naveen Srivastava, Hari Prasad, Mohini Polimetla, Phaneendra Chennupati, Eshwar Gupta, Mahesh Vykuntam, Sumantra Sarkar, Praveen Kumar Duvvuru, Theja Mardi, and G Swetha. 2024. 2.2 "Zen 4c": The AMD 5nm Area-Optimized ×86-64 Microprocessor Core. In *2024 IEEE International Solid-State Circuits Conference (ISSCC)*, Vol. 67. 38–40.

[12] Cadence. 2025. The Cadence Allegro X Design Platform. https://www.cadence.com/en_US/home/tools/pcb-design-and-analysis/allegro-x-design-platform.html.

[13] Qizhe Cai, Midhul Vuppalapati, Jaehyun Hwang, Christos Kozyrakis, and Rachit Agarwal. 2022. Towards us tail latency and terabit ethernet: disaggregating the host network stack. In *Proceedings of the ACM SIGCOMM 2022 Conference (SIGCOMM'22)*. 767–779.

[14] Xuzheng Chen, Jie Zhang, Ting Fu, Yifan Shen, Shu Ma, Kun Qian, Lingjun Zhu, Chao Shi, Yin Zhang, Ming Liu, et al. 2024. Demystifying datapath accelerator enhanced off-path smartnic. In *2024 IEEE 32nd International Conference on Network Protocols (ICNP'24)*. 1–12.

[15] Jack Choquette. 2022. Nvidia hopper gpu: Scaling performance. In *Proceedings of 2022 IEEE Hot Chips 34 Symposium (HCS)*. IEEE Computer Society, 1–46.

[16] Austin T. Clements, M. Frans Kaashoek, Nickolai Zeldovich, Robert T. Morris, and Eddie Kohler. 2013. The scalable commutativity rule: designing scalable software for multicore processors. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles (SOSP'13)*. 1–17.

[17] Ampere Computing. 2024. Breaking Boundaries: AmpereOne's Disaggregation Strategy for the Next-Gen Cloud. https://amperecomputing.com/blogs/next-gen-cloud.

[18] The UCIe Consortium. 2025. Universal Chiplet Interconnect Express (UCIe). https://www.uciexpress.org.

[19] David Culler, Richard Karp, David Patterson, Abhijit Sahay, Klaus Erik Schauser, Eunice Santos, Ramesh Subramonian, and Thorsten Von Eicken. 1993. LogP: Towards a realistic model of parallel computation. In *Proceedings of the fourth ACM SIGPLAN symposium on Principles and practice of parallel programming*.

[20] Arnaldo Carvalho De Melo. 2010. The new linux'perf'tools. In *Slides from Linux Kongress*, Vol. 18. 1–42.

[21] The devicetree.org community. 2025. The Devicetree Specification. https://www.devicetree.org.

[22] Daniel Firestone, Andrew Putnam, Sambhrama Mundkur, Derek Chiou, Alireza Dabagh, Mike Andrewartha, Hari Angepat, Vivek Bhanu, Adrian Caulfield, Eric Chung, Harish Kumar Chandrappa, Somesh Chaturmohta, Matt Humphrey, Jack Lavier, Norman Lam, Fengfen Liu, Kalin Ovtcharov, Jitu Padhye, Gautham Popuri, Shachar Raindel, Tejas Sapre, Mark Shaw, Gabriel Silva, Madhan Sivakumar, Nisheeth Srivastava, Anshuman Verma, Qasim Zuhair, Deepak Bansal, Doug Burger, Kushagra Vaid, David A. Maltz, and Albert Greenberg. 2018. Azure accelerated networking: SmartNICs in the public cloud. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI'18)*. 51–66.

[23] GigaIO. 2025. GigaIO SuperNode. https://gigaio.com/supernode/.

[24] Groq. 2025. GroqRack Compute Cluster. https://groq.com/groqrack/.

[25] Boris Grot, Joel Hestness, Stephen W. Keckler, and Onur Mutlu. 2009. Express Cube Topologies for on-Chip Interconnects. In *2009 IEEE 15th International Symposium on High Performance Computer Architecture*. 163–174.

[26] Zerui Guo, Jiaxin Lin, Yuebin Bai, Daehyeok Kim, Michael Swift, Aditya Akella, and Ming Liu. 2023. LogNIC: A High-Level Performance Model for SmartNICs. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'23)*. 916–929.

[27] Zerui Guo, Hua Zhang, Chenxingyu Zhao, Yuebin Bai, Michael Swift, and Ming Liu. 2023. LEED: A Low-Power, Fast Persistent Key-Value Store on SmartNIC JBOFs. In *Proceedings of the ACM SIGCOMM 2023 Conference (SIGCOMM'23)*. 1012–1027.

[28] NVIDIA H100. 2024. NVIDIA H100 Tensor Core GPU. https://www.nvidia.com/en-us/data-center/h100/.

[29] Yongchao He, Wenfei Wu, Yanfang Le, Ming Liu, and ChonLam Lao. 2023. A Generic Service to Provide In-Network Aggregation for Key-Value Streams. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'23), Volume 2*. 33–47.

[30] Wentao Hou, Jie Zhang, Zeke Wang, and Ming Liu. 2024. Understanding Routable PCIe Performance for Composable Infrastructures. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI'24)*. 297–312.

[31] Jingcao Hu and Radu Marculescu. 2003. Energy-aware mapping for tile-based NoC architectures under performance constraints. In *Proceedings of the 2003 Asia and South Pacific Design Automation Conference*. 233–239.

[32] Jaehyun Hwang, Midhul Vuppalapati, Simon Peter, and Rachit Agarwal. 2021. Rearchitecting linux storage stack for μs latency and high throughput. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI'21)*. 113–128.

[33] IDTechEx. 2025. Chiplet Technology 2025-2035: Technology, Opportunities, Applications. https://www.idtechex.com/en/research-report/chiplet-technology-2025-2035-technology-opportunities-applications/1041.

[34] Intel. 2024. Software-Defined Vehicle Transformation Starts with Intel. https://download.intel.com/newsroom/2024/automotive/Intel-SDV-Demo-Fact-Sheet.pdf.

[35] Sheng Jiang and Ming Liu. 2025. Building an Elastic Block Storage over EBOFs Using Shadow Views. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI'25)*. 1137–1153.

[36] Yuyuan Kang and Ming Liu. 2025. Understanding and Profiling NVMe-over-TCP Using ntprof. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI'25)*. 1117–1136.

[37] Keysight. 2024. What is a Chiplet, and Why Should You Care? https://www.keysight.com/blogs/en/tech/sim-des/2024/2/8/what-is-a-chiplet-and-why-should-you-care.

[38] John Kim, Wiliam J. Dally, Steve Scott, and Dennis Abts. 2008. Technology-Driven, Highly-Scalable Dragonfly Topology. In *Proceedings of the 35th Annual International Symposium on Computer Architecture*. 77–88.

[39] Praveen Kumar, Nandita Dukkipati, Nathan Lewis, Yi Cui, Yaogong Wang, Chonggang Li, Valas Valancius, Jake Adriaens, Steve Gribble, Nate Foster, and Amin Vahdat. 2019. Picnic: predictable virtualized nic. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM'19)*. 351–366.

[40] Collin Lee and John Ousterhout. 2019. Granular computing. In *Proceedings of the Workshop on Hot Topics in Operating Systems*. 149–154.

[41] Xiao Li, Zerui Guo, Yuebin Bai, Mehash Ketkar, Hugh Willkinson, and Ming Liu. 2025. Understanding and Profiling CXL.mem Using PathFinder. In *Proceedings of the ACM SIGCOMM 2025 Conference (SIGCOMM'25)*.

[42] Ming Liu. 2020. *Building Distributed Systems Using Programmable Networks*. University of Washington.

[43] Ming Liu. 2023. Fabric-Centric Computing. In *Proceedings of the 19th Workshop on Hot Topics in Operating Systems (HotOS'23)*. 118–126.

[44] Ming Liu, Tianyi Cui, Henry Schuh, Arvind Krishnamurthy, Simon Peter, and Karan Gupta. 2019. Offloading distributed applications onto smartNICs using iPipe. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM'19)*. 318–333.

[45] Ming Liu, Arvind Krishnamurthy, Harsha V. Madhyastha, Rishi Bhardwaj, Karan Gupta, Chinmay Kamat, Huapeng Yuan, Aditya Jaltade, Roger Liao, Pavan Konka, and Anoop Jawahar. 2020. Fine-Grained Replicated State Machines for a Cluster Storage System . In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI'20)*. 305–323.

[46] Ming Liu, Liang Luo, Jacob Nelson, Luis Ceze, Arvind Krishnamurthy, and Kishore Atreya. 2017. IncBricks: Toward In-Network Computation with an In-Network Cache. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'17)*. 795–809.

[47] Ming Liu, Simon Peter, Arvind Krishnamurthy, and Phitchaya Mangpo Phothilimthana. 2019. E3: Energy-Efficient Microservices on SmartNIC-Accelerated Servers. In *2019 USENIX Annual Technical Conference (USENIX ATC'19)*. 363–378.

[48] Liang Luo, Ming Liu, Jacob Nelson, Luis Ceze, Amar Phanishayee, and Arvind Krishnamurthy. 2017. Motivating in-network aggregation for distributed deep neural network training. In *Workshop on Approximate Computing Across the Stack*.

[49] Zhihong Luo, Sam Son, Sylvia Ratnasamy, and Scott Shenker. 2024. Harvesting Memory-bound CPU Stall Cycles in Software with MSH. In *Proceedings of 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI'24)*. 57–75.

[50] Nestor Maslej, Loredana Fattorini, Raymond Perrault, Yolanda Gil, Vanessa Parli, Njenga Kariuki, Emily Capstick, Anka Reuel, Erik Brynjolfsson, John Etchemendy, Katrina Ligett, Terah Lyons, James Manyika, Juan Carlos Niebles, Yoav Shoham, Russell Wald, Tobi Walsh, Armin Hamrah, Lapo Santarlasci, Julia Betts Lotufo, Alexandra Rome, Andrew Shi, and Sukrut Oak. 2025. Artificial Intelligence Index Report 2025. arXiv:2504.07139 [cs.AI] https://arxiv.org/abs/2504.07139

[51] A. Medina, N. Taft, K. Salamatian, S. Bhattacharyya, and C. Diot. 2002. Traffic matrix estimation: existing techniques and new directions. In *Proceedings of the 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM'02)*. 161–174.

[52] Jaehong Min, Ming Liu, Tapan Chugh, Chenxingyu Zhao, Andrew Wei, In Hwan Doh, and Arvind Krishnamurthy. 2021. Gimbal: enabling multi-tenant storage disaggregation on SmartNIC JBOFs. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference (SIGCOMM'21)*. 106–122.

[53] Jaehong Min, Chenxingyu Zhao, Ming Liu, and Arvind Krishnamurthy. 2023. eZNS: An Elastic Zoned Namespace for Commodity ZNS SSDs. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI'23)*. 461–477.

[54] Jaehong Min, Chenxingyu Zhao, Ming Liu, and Arvind Krishnamurthy. 2024. eZNS: Elastic Zoned Namespace for Enhanced Performance Isolation and Device Utilization. *ACM Trans. Storage* 20, 3, Article 16 (June 2024), 41 pages.

[55] Moscibroda, Thomas and Mutlu, Onur. 2009. A case for bufferless routing in on-chip networks. In *Proceedings of the 36th Annual International Symposium on Computer Architecture*. 196–207.

[56] Ashley O. Munch, Nevine Nassif, Carleton L. Molnar, Jason Crop, Rich Gammack, Chinmay P. Joshi, Goran Zelic, Kambiz Munshi, Min Huang, Charles R. Morganti, Sireesha Kandula, and Arijit Biswas. 2024. 2.3 Emerald Rapids: 5th-Generation Intel® Xeon® Scalable Processors. In *2024 IEEE International Solid-State Circuits Conference (ISSCC)*, Vol. 67. 40–42.

[57] Benjamin Munger, Kathy Wilcox, Jeshuah Sniderman, Chuck Tung, Brett Johnson, Russell Schreiber, Carson Henrion, Kevin Gillespie, Tom Burd, Harry Fair, David Johnson, Jonathan White, Scott McLelland, Steven Bakke, Javin Olson, Ryan McCracken, Matthew Pickett, Aaron Horiuchi, Hien Nguyen, and Tim H Jackson. 2023. "Zen 4": The AMD 5nm 5.7GHz x86-64 Microprocessor Core. In *2023 IEEE International Solid-State Circuits Conference (ISSCC)*. 38–39.

[58] Samuel Naffziger, Noah Beck, Thomas Burd, Kevin Lepak, Gabriel H. Loh, Mahesh Subramony, and Sean White. 2021. 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)*. 57–70.

[59] Nevine Nassif, Ashley O. Munch, Carleton L. Molnar, Gerald Pasdast, Sitaraman V. Lyer, Zibing Yang, Oscar Mendoza, Mark Huddart, Srikrishnan Venkataraman, Sireesha Kandula, Rafi Marom, Alexandra M. Kern, Bill Bowhill, David R. Mulvihill, Srikanth Nimmagadda, Varma Kalidindi, Jonathan Krause, Mohammad M. Haq, Roopali Sharma, and Kevin Duda. 2022. Sapphire Rapids: The Next-Generation Intel Xeon Scalable Processor. In *Proceedings of 2022 IEEE International Solid-State Circuits Conference (ISSCC)*, Vol. 65. 44–46.

[60] Rolf Neugebauer, Gianni Antichi, José Fernando Zazo, Yury Audzevich, Sergio López-Buedo, and Andrew W. Moore. 2018. Understanding PCIe performance for end host networking. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication (SIGCOMM'18)*. 327–341.

[61] The NextPlatform. 2023. AWS ADOPTS ARM V2 CORES FOR EXPANSIVE GRAVITON4 SERVER CPU. https://www.nextplatform.com/2023/11/28/aws-adopts-arm-v2-cores-for-expansive-graviton4-server-cpu/.

[62] NVIDIA. 2025. NVIDIA GB200 NVL72. https://www.nvidia.com/en-us/data-center/gb200-nvl72/.

[63] NVIDIA. 2025. NVIDIA HGX Platform. https://www.nvidia.com/en-us/data-center/hgx/.

[64] D.A. Patterson. 2006. RAMP: research accelerator for multiple processors - a community vision for a shared experimental parallel HW/SW platform. In *Proceedings of 2006 IEEE International Symposium on Performance Analysis of Systems and Software*.

[65] David Patterson, Thomas Anderson, Neal Cardwell, Richard Fromm, Kimberley Keeton, Christoforos Kozyrakis, Randi Thomas, and Katherine Yelick. 1997. Intelligent RAM (IRAM): Chips that remember

and compute. In *Proceedings of 1997 IEEE International Solids-State Circuits Conference. Digest of Technical Papers*. 224–225.

[66] David Patterson, Thomas Anderson, Neal Cardwell, Richard Fromm, Kimberly Keeton, Christoforos Kozyrakis, Randi Thomas, and Katherine Yelick. 2002. A case for intelligent RAM. *IEEE micro* 17, 2 (2002), 34–44.

[67] Phitchaya Mangpo Phothilimthana, Ming Liu, Antoine Kaufmann, Simon Peter, Rastislav Bodik, and Thomas Anderson. 2018. Floem: A Programming System for NIC-Accelerated Network Applications. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI'18)*. 663–679.

[68] The Next Platform. 2025. SILICON ONE G200 FINALLY DRIVES CISCO'S AI NETWORKING BUSINESS. https://www.nextplatform.com/2025/05/21/silicon-one-g200-finally-drives-ciscos-ai-networking-business/.

[69] The Next Platform. 2025. THE AI DATACENTER IS RAVENOUS FOR 102.4 TB/SEC ETHERNET SWITCH ASICS. https://www.nextplatform.com/2025/06/03/the-ai-datacenter-is-ravenous-for-102-4-tb-sec-ethernet/.

[70] Raghu Prabhakar and Sumti Jairath. 2021. SambaNova SN10 RDU: Accelerating software 2.0 with dataflow. In *2021 IEEE Hot Chips 33 Symposium (HCS)*. 1–37.

[71] Raghu Prabhakar, Sumti Jairath, and Jinuk Luke Shin. 2022. Sambanova sn10 rdu: A 7nm dataflow architecture to accelerate software 2.0. In *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, Vol. 65. 350–352.

[72] Raghu Prabhakar, Ram Sivaramakrishnan, Darshan Gandhi, Yun Du, Mingran Wang, Xiangyu Song, Kejie Zhang, Tianren Gao, Angela Wang, Xiaoyan Li, et al. 2024. Sambanova sn40l: Scaling the ai memory wall with dataflow and composition of experts. In *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 1353–1366.

[73] The Open Compute Project. 2021. OpenHBI Specification Version 1.0. https://www.opencompute.org/documents/odsa-openhbi-v1-0-spec-rc-final-1-pdf.

[74] The Open Compute Project. 2022. Bunch of Wires PHY Specification. https://www.opencompute.org/documents/bunch-of-wires-phy-specification-pdf.

[75] Yiming Qiu, Qiao Kang, Ming Liu, and Ang Chen. 2020. Clara: Performance Clarity for SmartNIC Offloading. In *Proceedings of the 19th ACM Workshop on Hot Topics in Networks (HotNets'20)*. 16–22.

[76] Yiming Qiu, Jiarong Xing, Kuo-Feng Hsu, Qiao Kang, Ming Liu, Srinivas Narayana, and Ang Chen. 2021. Automated SmartNIC Offloading Insights for Network Functions. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles (SOSP'21)*. 772–787.

[77] Sivasankar Radhakrishnan, Yilong Geng, Vimalkumar Jeyakumar, Abdul Kabbani, George Porter, and Amin Vahdat. 2014. SENIC: Scalable NIC for End-Host rate limiting. In *Proceedings of 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI'14)*. 475–488.

[78] Albert Reuther, Peter Michaleas, Michael Jones, Vijay Gadepally, Siddharth Samsi, and Jeremy Kepner. 2022. AI and ML accelerator survey and trends. In *2022 IEEE High Performance Extreme Computing Conference (HPEC)*. 1–10.

[79] Zhenyuan Ruan, Tong He, and Jason Cong. 2019. INSIDER: Designing In-Storage computing system for emerging High-Performance drive. In *Proceedings of 2019 USENIX Annual Technical Conference (USENIX ATC'19)*. 379–394.

[80] Ahmed Saeed, Nandita Dukkipati, Vytautas Valancius, Vinh The Lam, Carlo Contavalli, and Amin Vahdat. 2017. Carousel: Scalable traffic shaping at end hosts. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM'17)*. 404–417.

[81] Henry N. Schuh, Weihao Liang, Ming Liu, Jacob Nelson, and Arvind Krishnamurthy. 2021. Xenic: SmartNIC-Accelerated Distributed Transactions. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles (SOSP'21)*. 740–755.

[82] Abu Sebastian, Manuel Le Gallo, Riduan Khaddam-Aljameh, and Evangelos Eleftheriou. 2020. Memory devices and applications for in-memory computing. *Nature nanotechnology* 15, 7 (2020), 529–544.

[83] Naveen Kr. Sharma, Ming Liu, Kishore Atreya, and Arvind Krishnamurthy. 2018. Approximating Fair Queueing on Reconfigurable Switches. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI'18)*. 1–16.

[84] Naveen Kr. Sharma, Chenxingyu Zhao, Ming Liu, Pravein G Kannan, Changhoon Kim, Arvind Krishnamurthy, and Anirudh Sivaraman. 2020. Programmable Calendar Queues for High-speed Packet Scheduling . In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI'20)*. 685–699.

[85] Teja Singh, Spence Oliver, Sundar Rangarajan, Shane Southard, Carson Henrion, Alex Schaefer, Brett Johnson, Sarah Bartaszewicz Tower, Kathy Hoover, Deepesh John, Ted Antoniadis, Shravan Lakshman, Vibhor Mittal, Brian Kasprzyk, Ross McCoy, Kurt Mohlman, Anitha Mohan, Hon-Hin Wong, Daryl Lieu, Russell Schreiber, Sahilpreet Singh, Nick Lance, Darryl Prudich, Justin Coppin, Tim Jackson, Anita Karegar, Ryan Miller, Sabeesh Balagangadharan, James Pistole, Wilson Li, and Michael McCabe. 2025. "Zen 5": The AMD High-Performance 4nm x86-64 Microprocessor Core. In *Proceedings of 2025 IEEE International Solid-State Circuits Conference (ISSCC)*, Vol. 68. 1–3.

[86] Teja Singh, Sundar Rangarajan, Deepesh John, Russell Schreiber, Spence Oliver, Rajit Seahra, and Alex Schaefer. 2020. 2.1 Zen 2: The AMD 7nm Energy-Efficient High-Performance x86-64 Microprocessor Core. In *2020 IEEE International Solid-State Circuits Conference - (ISSCC)*. 42–44.

[87] Alan Smith, Eric Chapman, Chintan Patel, Raja Swaminathan, John Wuu, Tyrone Huang, Wonjun Jung, Alexander Kaganov, Hugh McIntyre, and Ramon Mangaser. 2024. 11.1 AMD InstinctTM MI300 Series Modular Chiplet Package – HPC and AI Accelerator for Exa-Class Systems. In *Proceedings of 2024 IEEE International Solid-State Circuits Conference (ISSCC)*, Vol. 67. 490–492.

[88] Armando Solar-Lezama, Christopher Grant Jones, and Rastislav Bodik. 2008. Sketching concurrent data structures. In *Proceedings of the 29th ACM SIGPLAN Conference on Programming Language Design and Implementation*. 136–148.

[89] Synopsys. 2025. What are Chiplets. https://www.synopsys.com/glossary/what-are-chiplets.html.

[90] Microchip USA. 2023. What is a Chiplet? https://www.microchipusa.com/industry-news/what-is-a-chiplet.

[91] Raj R. Varada, Rohini Krishnan, Ajith Subramonia, Rathish Chandran, Kalyana Chakravarthy, Uttpal D. Desai, Sumedha Limaye, Puneesh Puri, David R. Mulvihill, Mike Bichan, Martin Koolhaas, Vijayalakshmi Ramachandran, and Srinivasu Kendle. 2025. 2.3 Granite Rapids-D: Intel Xeon 6 SoC for vRAN, Edge, Networking, and Storage. In *Proceedings of 2025 IEEE International Solid-State Circuits Conference (ISSCC)*, Vol. 68. 48–50.

[92] Yehuda Vardi. 1996. Network tomography: Estimating source-destination traffic intensities from link data. *J. Amer. Statist. Assoc.* 91, 433 (1996), 365–377.

[93] Jasmina Vasiljevic and Davor Capalija. 2024. Blackhole & tt-metalium: The standalone ai computer and its programming model. In *2024 IEEE Hot Chips 36 Symposium (HCS)*. 1–30.

[94] Naveen Verma, Hongyang Jia, Hossein Valavi, Yinqi Tang, Murat Ozatay, Lung-Yen Chen, Bonan Zhang, and Peter Deaville. 2019. In-memory computing: Advances and prospects. *IEEE solid-state circuits magazine* 11, 3 (2019), 43–55.

[95] Midhul Vuppalapati, Saksham Agarwal, Henry Schuh, Baris Kasikci, Arvind Krishnamurthy, and Rachit Agarwal. 2024. Understanding the Host Network. In *Proceedings of the ACM SIGCOMM 2024 Conference*. 581–594.

[96] John Wawrzynek. 2015. *Accelerating Science Driven System Design With RAMP*. Technical Report. Univ. of California, Berkeley, CA (United States).

[97] Zhewei Wei, Ge Luo, Ke Yi, Xiaoyong Du, and Ji-Rong Wen. 2015. Persistent data sketching. In *Proceedings of the 2015 ACM SIGMOD international conference on Management of Data*. 795–810.

[98] WikiChip. 2025. Chiplet. https://en.wikichip.org/wiki/chiplet.

[99] Wikipedia. 2025. 2.5D integrated circuit. https://en.wikipedia.org/wiki/2.5D_integrated_circuit.

[100] Wikipedia. 2025. Advanced packaging (semiconductors). https://en.wikipedia.org/wiki/Advanced_packaging_(semiconductors).

[101] Samuel Williams, Andrew Waterman, and David Patterson. 2009. Roofline: an insightful visual performance model for multicore architectures. *Commun. ACM* 52, 4 (2009), 65–76.

[102] Xincheng Xie, Wentao Hou, Zerui Guo, and Ming Liu. 2025. Building Massive MIMO Baseband Processing on a Single-Node Supercomputer. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI'25)*. 1221–1242.

[103] Tong Yang, Lingtong Liu, Yibo Yan, Muhammad Shahzad, Yulong Shen, Xiaoming Li, Bin Cui, and Gaogang Xie. 2017. Sf-sketch: A fast, accurate, and memory efficient data structure to store frequencies of data items. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*. 103–106.

[104] Jie Zhang, Hongjing Huang, Xuzheng Chen, Xiang Li, Jieru Zhao, Ming Liu, and Zeke Wang. 2025. RpcNIC: Enabling Efficient Datacenter RPC Offloading on PCIe-attached SmartNICs. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA'25)*. 1379–1394.

[105] Chenxingyu Zhao, Tapan Chugh, Jaehong Min, Ming Liu, and Arvind Krishnamurthy. 2022. Dremel: Adaptive Configuration Tuning of RocksDB KV-Store. *Proc. ACM Meas. Anal. Comput. Syst.* 6, 2, Article 37 (June 2022), 30 pages.

[106] Chenxingyu Zhao, Jaehong Min, Ming Liu, and Arvind Krishnamurthy. 2025. White-Boxing RDMA with Packet-Granular Software Control. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI'25)*. 427–449.

[107] Pengfei Zuo, Huimin Lin, Junbo Deng, Nan Zou, Xingkun Yang, Yingyu Diao, Weifeng Gao, Ke Xu, Zhangyu Chen, Shirui Lu, et al. 2025. Serving Large Language Models on Huawei CloudMatrix384. *arXiv preprint arXiv:2506.12708* (2025).