CXL and the Return of Scale-Up Database Engines

Alberto Lerner
eXascale Infolab
University of Fribourg, Switzerland
alberto.lerner@unifr.ch

Gustavo Alonso
Systems Group, Department of Computer Science
ETH Zurich, Switzerland
alonso@inf.ethz.ch

ABSTRACT

The trend toward specialized processing devices such as TPUs, DPUs, GPUs, and FPGAs has exposed the weaknesses of PCIe in interconnecting these devices and their hosts. Several attempts have been proposed to improve, augment, or downright replace PCIe, and more recently, these efforts have converged into a standard called Compute Express Link (CXL). CXL is already on version 2.0 in terms of commercial availability, but its potential to radically change the conventional server architecture has only just started to surface. For example, CXL can increase the bandwidth and quantity of memory available to any single machine beyond what that machine can originally provide, most importantly, in a manner that is fully transparent to software applications.

We argue, however, that CXL can have a broader impact beyond memory expansion and deeply affect the architecture of data-intensive systems. In a nutshell, while the cloud favored *scale-out* approaches that grew in capacity by adding full servers to a rack, CXL brings back *scale-up* architectures that can grow by fine-tuning individual resources, all while transforming the rack into a large shared-memory machine. In this paper, we describe why such architectural transformations are now possible, how they benefit emerging heterogeneous hardware platforms for data-intensive systems, and the associated research challenges.

1 INTRODUCTION

The availability of elastic hardware in the cloud has enabled new, ever-demanding classes of applications to emerge. Many of these applications benefit from the acceleration provided by *heterogeneous computing devices*, such as Smart NICs, FPGAs, GPUs, TPUs, and DPUs, to name the most common ones [54]. These devices are often more powerful than the CPU for specific tasks [35, 62], but they expose the fact that the most currently used system interconnect, the PCIe system [43], is unsuitable for this purpose [32, 59]. The bandwidth and latency incurred by moving data back and forth between CPU and devices erodes the devices' benefits and forces applications to batch data transfers and restructure themselves to hide the transfer latency. Not surprisingly, recent devices are packed with memory to hold as much data as possible locally and avoid

data movements through the PCIe system. The situation reached the point that in some computing devices, 75% of the power and 80% of their area is consumed by memory [14, 15].

In response, hardware vendors have proposed a different approach to device integration. Rather than copying data in and out of a device, the approach interconnects different accelerators and CPUs through distributed shared memory. This gives CPUs and devices the ability to cache one another's memory coherently [51]. Of course, the local memory size is still relevant, but with coherence, the application need not control how data moves; it simply accesses data as if it were local, and the added machinery efficiently relocates data as necessary. Because coherency is mediated through hardware, which is designed to manipulate small data units, coherent transfers can achieve latency and bandwidth only moderately worse than what a CPU can achieve. Several competing proposals ensued that allowed this type of memory-based integration to become public standards, such as OpenCAPI [40], CCIX [13], and Gen-Z [18]. These efforts have now converged into the *Compute Express Link* standard (CXL) [49].

CXL is a strict superset of PCIe, which thereby fosters easier adoption. It is, however, much more than just an interconnect protocol. It consists of three classes of interfaces. First, the *I/O interface* increases the speed of PCIe without major changes to its semantics, i.e., it can still copy memory regions from one device to another in a non-coherent manner. Second, the *memory interface* allows a host CPU to coherently access the memory or storage of peripheral devices. Lastly, the *cache interface* allows peripheral devices to coherently access and cache data from the host's memory.

The architectural change that the two latter interfaces represent was almost unimaginable just a few years ago. Cache coherency protocols have existed for as long as caches have, but CPU manufacturers treated their coherency protocols as trade secrets. For this reason, coherency could only be established across a vendor's CPU and its memory controllers. With CXL, different CPUs can talk to different controllers, including those based on computing devices, because the protocol for doing so is now an open standard.

As big as this change is, it is not the only benefit CXL brings. In its latest version, the standard goes beyond the traditional role of an interconnect; it becomes an alternative to a rack-level networking fabric that is both more performant than current Ethernet-based systems as well as more expressive. It offers additional features such as memory sharing and trusted memory enclaves, as opposed to just exposing a packet sending and receiving interface as in TCP/IP, or a remote memory access interface as in RDMA.

In this paper, we focus on these capabilities of CXL and argue that they enable *scale-up* (i.e., vertically scaled) database engines that significantly differ from the *scale-out* (i.e., horizontally scaled) systems that have dominated the landscape in the last years due to the constraints imposed by cloud architectures.

In what follows, we first provide a brief introduction to CXL (Sec. 2) and then present in detail three new architectural possibilities enabled by novel CXL features (Sec. 3), discussing the challenges that each brings. We show that these architectures can further benefit from Near-Data Processing capabilities (Sec. 4) and by Heterogeneous Processing ones (Sec. 5). Lastly, we also discuss related efforts (Sec. 6) and close with our conclusions (Sec. 7).

2 BACKGROUND AND MOTIVATION

CXL is, in essence, a set of new protocols to connect hosts to peripheral devices. It helps to start our discussion by revisiting how several cores in a given CPU are connected.

2.1 Multicores and Memory Coherency

A multicore server is a distributed system where CPU cores share the server's memory. Each core can cache data in small blocks, called *cache lines*, occasionally duplicating that data. Data consistency issues may arise because two cores may wish to modify the same line simultaneously. *Cache coherence* avoids the issues by maintaining two invariants: (1) writes to the same memory location are serialized; and (2) every write is eventually made visible to all cores [51]. Simply put, when a core intends to modify a cached line, it triggers a process known as cache invalidation, ensuring that only that copy of the line remains cached.

Cache coherence is handled by hardware components, usually via cache and memory controllers, as Figure 1(a) depicts. Until recently, the only entities that could cache lines were CPU cores, and the memory from which they could cache was limited to a server's available DRAM. In this scenario, we say that the *coherency domain* extends only to the CPU.

2.2 Coherent and Non-Coherent Data Transfers

Without CXL, data is copied between servers and devices through PCIe exchanges called *transactions*. PCIe transactions are not coherent because the device is outside the reach of the cache invalidation messages. Data copies can quietly become stale.

CXL supports extended coherence domains by creating new types of transactions between servers and devices. It does so in a backward-compatible way. The PCIe transactions are wrapped in a sub-protocol called `cxl.io` and two other sub-protocols are added: `cxl.mem` and `cxl.cache`. The server uses the former to communicate with the devices's memory controller as if it resided locally on its motherboard. The device uses the latter sub-protocol to cache contents managed by the server's directory controller. Figure 1(b) depicts the two new protocols. Because of the coherent sub-protocols, any device's memory can be seamlessly incorporated into the overall system, and the device can cache data that lives in the system's memory. Devices are able to use one or both of the new sub-protocols. If they only use `cxl.cache` and do not contribute memory to the system, they are called *Type 1* devices. If they only use `cxl.mem` and do not cache memory from the host, they are called *Type 3*. Devices that use both are known as *Type 2*.

2.3 CXL Status and Availability

CXL is a public consortium led by Intel, and it has had four major releases so far. The 1.1 version focuses on local memory expansion,

allowing a server to access more memory than is available in its DIMM slots through locally attached CXL devices called *memory expanders*. Release 2.0 introduced basic forms of interconnects (i.e., CXL switches). It allows memory expander devices to be installed in remote chassis and be shared by different servers. As of this writing, two generations of Intel CPUs support CXL 1.1 [22, 23], and a new generation supporting 2.0 was just released [24]. There are manufacturers offering Type 3 expanders, including some for CXL 2.0 (e.g., [36, 46, 47]). The first CXL switch silicons and adapters are also becoming available [45, 58]. Most importantly, some established database vendors are evaluating these technologies [9, 27].

CXL versions 3.0 and 3.1 support more sophisticated networking and additional memory-sharing modes, including peer-to-peer exchanges among devices. These versions depend on improvements in PCIe, which will be accomplished in the already ratified PCIe Gen 6 and the Gen 7 [43]. We expect the next generation of Intel and AMD CPUs to support PCIe 6 and the newest CXL versions.

2.4 CXL Performance Characterization

It is easy to look into CXL memory performance by comparing it to NUMA memory access. Current conventional NUMA servers typically comprise two sockets, each containing a CPU and half of the system's DRAM. When a CXL memory expander is used, it effectively attaches more DRAM DIMMs to the system by creating an additional NUMA node, albeit one without any cores. We note, however, that simply modeling CXL memory as NUMA can be inaccurate. The number of memory controllers in an expander and, thus, available bandwidth varies from product to product. Moreover, the type of memory used in the expander need not be the same as the one in the hosts.

Preliminary benchmarks by Microsoft, Meta, and Intel about CXL memory performance are already available (resp., [31, 34, 52]). The perceived latency for CXL memory access is slightly higher than that for remote NUMA access, roughly in the 200–400 nanoseconds range. As noted above, the bandwidth highly depends on the expander's characteristics. It can be lower or higher than that of a memory controller found within a host NUMA node. As just mentioned, nothing prevents an expander from using HBM instead of DDR memory. We summarize the performance results of the three papers we mention next.

The micro-benchmarks from Intel investigate both latency and bandwidth [52]. Regarding latency, executing a load instruction against a given type of CXL-attached memory can take just 35% longer than the equivalent NUMA memory access. Executing a store under the same conditions can present slightly lower but equivalent overheads. Regarding bandwidth, transfers from NUMA nodes can be 70% efficient when considering only loads, compared to 46% efficiency through a CXL interconnect.

Meta published results on the end-to-end impact of using CXL in applications [34]. In that work, CXL memory stored cold pages, with the operating system swapping pages back and forth between the host DRAM and the CXL memory. In essence, CXL memory was used as an additional memory tier between the DRAM and storage. The results suggest that the bandwidth available from CXL memory is around 64 GB/s with latency only slightly larger than that of NUMA memory.
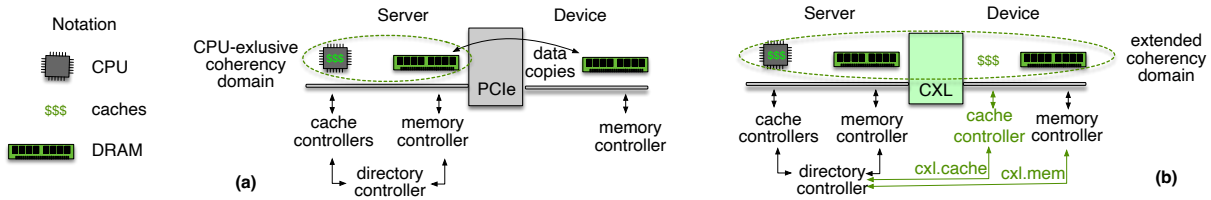
Figure 1: (a) Peripherals connected using PCIe cards are outside the coherency domain, even if they contain memory. (b) CXL is a set of protocols that allow components on the peripheral to join the CPU coherency domain.

Microsoft also performed an end-to-end study on the impact of CXL memory for their cloud environment [31]. While the results differ from workload to workload, the study found that under the expected latency increases, some 26% of the 158 workloads studied showed less than 1% performance penalty, and an additional 17% showed less than 5%. For database workloads, specifically TPC-H, the overheads were highly query-dependent but were mostly below 20%. This analysis already considers CXL switches and shared memory among a large number of machines in a rack.

2.5 CXL as Networking

Other studies suggest that CXL interconnects, the fabric carrying coherency traffic between servers and devices, can be significantly more efficient than RDMA-enabled networks [19, 25, 49]. As seen above, the latency of CXL communications averages in the low hundreds of nanoseconds, while the fastest RDMA exchanges take a few microseconds—a difference of at least 2.5×. The advantages go beyond latency. These studies also hint at bandwidth differences and partly attribute them to the fact that traditional network interface cards (NICs) underutilize the PCIe lanes they occupy. For instance, a 400 Gbps NIC (50 GB/s) uses 16 PCIe Gen5 lanes that, in the aggregate, can offer 64 GB/s [37]. This discrepancy means that over 20% of the available PCIe bandwidth does not translate into network bandwidth. CXL adapters utilize the full bandwidth.

From the application point of view, replacing RDMA with CXL turns an application that is networking API-based into a many-core, shared-memory application. This application can scale by launching additional threads and can access remote memory without incorporating networking APIs, such as Infiniband Verbs. Moreover, much of the memory access coordination is managed in hardware, through coherence, instead of by software. That is because CXL memory is accessed using the same familiar `load` and `store` instructions that applications use to access local memory. The hardware makes the remote access look transparent to the software, including maintaining coherence.

Ultimately, CXL's advantages go beyond software simplicity and performance. There are many functional aspects that RDMA simply cannot handle. RDMA does not provide a path from the CPU to accelerators (computing devices) for instance; CXL does. RDMA does not provide a path between different server components, for instance, from a NIC to an SSD; CXL does. RDMA only enables access to disaggregated memory; CXL allows disaggregation of an entire rack without requiring that the devices support anything other than PCIe at the physical level.

2.6 CXL Scalability and Fault Tolerance

CXL imposes a limit on the number of devices that contribute and/or cache memory from each other. Currently, a CXL coherence domain can support a *diameter* of up to 4096 devices. Arguably, this number of devices is amply sufficient to cover one rack—or even a small number of racks— especially because not all the devices need to be coherent. We note that such interconnection size limitations are not uncommon. For instance, RDMA requires the underlying Ethernet network to be lossless, something that it is also difficult to do at a large scale [20]. We expect that the CXL limitations will diminish as it evolves, as is typical with emerging technologies.

Failures can, of course, occur on a disaggregated rack, whether using RDMA or CXL. CXL does not add failure modes to those that are normally expected in a distributed system. However, memory expansion through CXL brings at least two advantages in exceptional scenarios. First, failure detection and propagation are built into the protocol through a set of mechanisms known as RAS (reliability, availability, and serviceability) [4]. Because the hardware is conceived to identify and address failures, the reaction times in a CXL platform are likely faster than in a traditional distributed system. Second, using CXL disaggregated memory instead of a remote machine's memory is a better scenario for failure probability due to the lower number of components and lower potential for load interference issues.

3 SHARED-MEMORY ARCHITECTURES

In this section, we will explore how CXL can expand a system's available memory in at least three ways. Figure 2 illustrates the resulting architectures. We will introduce each, discussing their implications, potential innovative applications, and the necessary research work to support them.

3.1 Memory Expansion

Database systems maintain a memory pool that all threads use to process queries or transactions called *Buffer Cache* or *Buffer Pool*. In disk-based systems, the pool is an intermediate stage and cache between persistent storage and each thread's private space [21]. In main-memory systems, the pool holds data in its entirety while still providing memory for query processing [17]. The most straightforward way to expand a database system's memory is to have the pool also manage CXL memory, as Figure 2(a) shows. After adding an expansion card to a system, the pool can access CXL memory using the same instructions it uses for conventional memory, no API needed, while the hardware takes care of coherency.
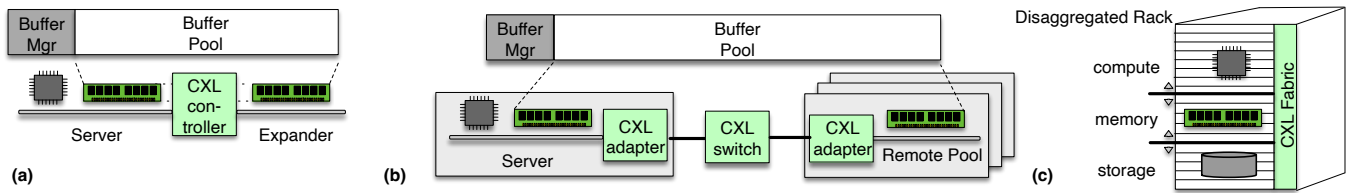
Figure 2: CXL allows different memory expansion architectures: (a) local expansion and (b) far-memory pooling, and (c) full-rack disaggregation. The latter two are achieved through specialized CXL switches.

However, a database system now has to match the different data placement options to the requirements for data access. This mapping should be flexible enough to accommodate different types of memory expanders and dynamic enough to reflect a system's instantaneous workload. Historically, this mapping was solved by a *memory tiering* approach. Hot pages are placed in fast-tier memory, and cold pages are in less-performing tiers. A system such as an OS paging mechanism would track pages' *temperatures* and make data placement decisions accordingly. Some, like Meta mentioned above [34], advocate a similar approach for CXL memory. Since it is long established that a database engine can better calculate the utility of keeping a page in a given memory tier than the OS [11], we posit that existing I/O optimization and buffer management techniques that have been successful in other deep memory hierarchy scenarios can play a new role here [33, 50].

A "killer app" for this architecture may lie in the very memory diversity it supports. In other words, the memory tiers can be carefully designed. CXL memory expanders not only break the notorious memory wall [33] but also rid the system of inter-dependencies. The expansion memory type need not match the memory type the current CPU/motherboard supports. The expansion can contain regions with slower/cheaper or faster/more expensive memory than the CPU at the system architect's discretion, even enabling the recycling of DRAM from older generations, an aspect with the potential to result in significant cost savings and a reduction of the environmental footprint of computers. From a database perspective, an interesting configuration would place the transactional workload on the local DRAM and use CXL memory for the analytic part. The data structures in the CXL memory could be specialized ones, such as data cubes, materialized tables, and denormalized tables, to cite a few. Thanks to the data placement, the OLTP and OLAP data structures would not interfere with each other. Exercising this flexibility allows fine-tuning the memory characteristics (and cost) on which a given database system thrives.

Efficiently expanding single-machine database systems with CXL memory raises several other important research questions:

- Are memory expanders fast enough for OLTP or will they be suitable mainly for OLAP? Can they be used to perform both on the same machine and what are the implications?
- Are the data structures kept in CXL memory the same as the ones that are successfully used in conventional memory? If not, are there data structures suitable for the increased non-uniformity in memory accesses that CXL memory creates?
- Should data structures span conventional and CXL memory? Could and should these structures be adaptable and offer dynamic data placement according to the access patterns?

- Alternatively, should we forego designing data structures and use memory expanders as paging block devices for integration simplicity, or would specialized allocating techniques benefit from byte-addressable memory?

3.2 Memory Pooling

Unfortunately, increasing individual servers' memory with expanders in a cloud setting can create a phenomenon called *stranded memory*. The memory physically present in one server may be underutilized by the systems running on it. Hyperscalers have reported that, as memory is one of the most expensive components in data centers, stranding is a major source of inefficiencies [31].

Recently, researchers have explored ways to reassign such stranded memory [2, 26, 57, 61]. The ideas have led to the notions of *far memory*, *remote memory*, and *disaggregated memory*. Far memory is a term typically used to refer to any generic configuration where the additional DRAM is not local. Remote memory is also far but generally refers to utilizing unused memory of other machines. Disaggregated memory is not allocated to any machine but is available for any servers to use. In most cases, the connection between server and memory is made through RDMA and the remote memory is treated as a block device with a paging mechanism to move data back and forth between the host and the far memory.

With CXL, a new architecture is possible that uses large, independent memory pools that can be carved into smaller pieces. Each server can access one or more of these pieces through CXL switches, and a single such memory pool can serve multiple servers simultaneously this way. Figure 2(b) depicts such a scenario. As discussed above, one of CXL's main innovations is its ability to assign a portion of a pool to a server without requiring its applications to access it through a non-coherent networking API. Direct, coherent memory access is far more efficient than what can be achieved in a distributed system that requires an external memory access API and leaves coherency to the application. Moreover, by cascading multiple CXL switches, CPUs on different machines can access a central memory expander containing a pool of memory available to all machines in a rack, very much like disaggregated storage is available in the cloud.

We posit that this architecture's killer app is making database systems better cloud denizens. Traditional databases assume that large amounts of data are loaded into a VM memory because the database performance depends on limiting I/O. The resulting VM is not elastic because databases do not cope well with dynamically varying the size of their buffer pools. For the same reason, database systems also miss the benefits of serverless technologies. The cost of quickly loading and unloading the state is too high.

Disaggregated memory implemented through CXL allows for placing the buffer pool on the disaggregated memory and using the local DRAM for query processing. If more query processing capacity is needed, new engines can be spawned and connected to the disaggregated memory so that these engines are immediately ready to run queries, as there is no need to warm up the database.

The additional latency of CXL memory plays only a minor role in databases [31] and is a good trade-off for far more elasticity than is feasible with engines where the data resides in local memory. Similarly, a database engine can be easily migrated when the buffer pool is in disaggregated memory. If the data structures and state of the engine itself are maintained in disaggregated memory, then migrating the entire engine to another machine becomes a far simpler operation.

From a research perspective, more interesting questions arise:

- Implementing cloud-friendly features requires rethinking the internal database architecture to remove the assumption that everything is in local memory. Extensive experimentation of what needs to be local and what can be placed in disaggregated memory will be required to determine which part of the engine can tolerate the additional latency of CXL memory. We discussed some of these architectural changes elsewhere [29].
- Assuming database systems become more elastic using the above techniques, should the granularity to consider be at the entire engine level, or can the elasticity be pushed down to the level of threads running queries?
- Threads running queries could be moved from machine to machine by keeping their state and working space in disaggregated memory. Alternatively, they can be created as the workload dictates. How would an engine operate under such a dynamically changing multiprogramming level?

3.3 Memory Sharing

We have seen how CXL can increase the memory availability and flexibility of single-server database systems. The next question to arise is whether CXL can also benefit larger database systems.

Traditionally, these systems scale by adopting distributed database techniques [1, 41, 56]. Simply applying the known scalability techniques into a CXL environment is unlikely to succeed for numerous reasons: to cite only a few, cache invalidation now crosses machines; updates imply more-onerous distributed locks; and memory is wasted as the same data is copied into the local buffer cache of several machines. Some of these issues were attenuated in the past through a mix of replication and sharding, often using RDMA to reduce the network overhead [6, 7, 53, 60]. However, these techniques invariably impose a notoriously fragile balance between consistency, symmetry in the system (e.g., with read-only copies), and data placement to minimize data movement, thereby limiting architectural freedom (e.g., [55]).

In contrast, CXL can integrate multiple servers and closely-placed large resource pooling modules without any of the above restrictions. The pooling modules can contain homogeneous DRAM devices but also devices with different mixes of volatile and non-volatile memory (e.g., [48]), creating storage pooling modules. Advanced CXL features such as *Global Integrated Memory* (GIM) and *Global Fabric-Attached Memory Devide* (GFAM), allow servers to share resources in the pooling modules collectively [8]. Simply put, the entire rack becomes a single shared-memory machine. Figure 2(c) depicts this scenario within one rack, but we believe the same features could also support spanning a small number of racks.

A killer app for this architecture is a larger database system in a truly scaled-up manner. Each thread in this database could run on a separate compute module but share the same memory and storage map with threads running in another computer module.

A fully disaggregated system like the one CXL enables raises yet unexplored scalability questions, such as:

- Hashing and sorting are at the core of most relational data processing, but it is not obvious how they would work at a rack-level scale. It is possible that accepted wisdom regarding when to use each one will change in a scale-out architecture.
- Given that each core can now access one to two orders of magnitude more memory than before, are the data structures we use to organize and index the data still effective at these new scales? Since the invalidation messages can now dominate access time, how is the *coherency traffic* generated by a typical data structure?
- Assuming we now have the freedom to engage a tremendous amount of resources to solve individual query operators, how do we schedule the machine resources across competing queries?

Rack-level integration is arguably the most impactful change enabled by CXL. It moves the memory unification effort to the hardware and thereby liberates the database system from managing coherence across servers. Moreover, CXL can attain bandwidth levels larger than those attained by today's networks and with far lower latency. We expect these possibilities alone to radically change the design of scalable databases and data processing engines, but there are two other profound possibilities. We describe the first in the next section and the second in the subsequent one.
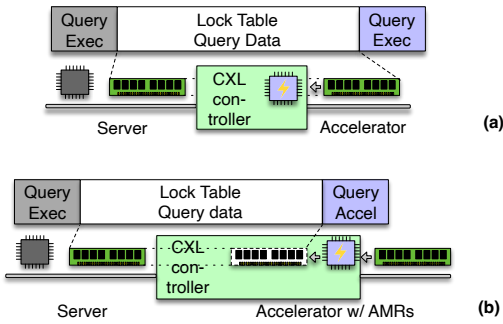
4 NEAR-DATA PROCESSING

CXL enables another important feature that applies to all the architectures we just described. To understand this feature, it helps to examine how a CXL device is built. A component in the device called coherency controller implements the CXL protocol. In a memory expander, this controller wraps existing DRAM DIMMs and intercepts all the requests to memory. Similarly, in a device that caches CXL memory, the controller maintains the cache state and responds to invalidation traffic. Put differently, the CXL controller is an independent entity from the memory it manages.

This independence opens a tremendous opportunity: The controller can be co-opted to perform computations over the data it transports. This ability is known as *near-data processing* (NDP). Figure 3(a) shows a controller capable of performing query processing on the device's side of the setup.

One may argue that NDP is hardly new. There are examples where a small processor is placed near memory [16] or in disaggregated memory [26] to accelerate query processing in research prototypes. Some products, such as the Oracle's retired SPARC S7, featured Data Analytics Accelerators (DAX) to support the offloading of basic relational operators [42]. However, in traditional NDP, query processing work is divided between the CPU and the accelerator because the accelerators do not support coherence. If one side changes the data, the other will not see it.

Figure 3: Near-data processing under CXL: (a) A processor or FPGA managing the expanded memory can be co-opted to execute a portion of a query. (b) A unique opportunity for acceleration exists through *active memory regions* (AMR).

In contrast, CXL-supported NDP allows both devices to work in parallel, not only because both can access data but also because the database system lock table can be shared, allowing each side to coordinate with the other. Being able to parallelize work that has been offloaded allows us to design a new generation of near-data accelerators that are more flexible and, most likely, faster.

A killer app for CXL-supported NDP is a new take on virtualizing memory. Since the controller intercepts all CXL traffic, it can implement an *active memory region* that is not backed by DRAM. Instead, the region corresponds to a computation that is executed when its memory addresses are accessed. If the computation is a streaming one, the results need not be materialized and could be fed to the application as it reads memory. Figure 3(b) illustrates such a use case. In databases, this can be used for on-the-fly implementations of view materialization and data transpositions [26] or better integration databases and smart storage systems [28].

CXL acceleration is not without challenges, some of which we list below:

- What portions of query processing should be done near the data? Some examples, such as compression and decompression, encryption and decryption, selection, projection, and filtering with LIKE predicates, have brought substantial advantages in practice [5, 10, 16, 26]. We believe other opportunities still exist.
- It is not inconceivable that CXL can support improving other fundamental mechanisms that are central to OLTP, such as collective communication, locking, timestamps, and dynamic data relocation across the system, among others.
- Many near-data processing accelerators have memory hierarchies as well. Should we treat them as local memory or use them as CXL memory, and in which proportion?

5 HETEROGENEOUS ARCHITECTURES

We have depicted in Figure 2(c) how to use CXL to connect pools of different resources, such as memory or storage. The same techniques allow other types of resources, such as FPGAs, GPUs, TPUs, and DPUs, to be similarly pooled and integrated into a rack-scale computer. For instance, as we just discussed, an accelerated CXL controller may offer services based on those resources through an active memory range.

This composability possibility opens a research field of its own:

- What should a heterogeneous machine look like now that different processing devices can be independently pooled? How should we select devices and balance resources across them?
- Can we and should we build machines tailored to specific workloads? For instance, machine learning (ML) is taxing database engines because the data often has to be taken out of the database to run it through ML tools. A heterogeneous architecture that seamlessly integrates CPUs and GPUs makes it possible to implement ML operators directly on the database engine.

6 RELATED EFFORTS

CXL is the result of consolidating several projects that came before it, most notably CCIX [13], GenZ [18], and OpenCAPI [40]. An overwhelming number of institutions and companies are working together to advance the protocol [12]. The consolidation, however, has not been complete. Some proprietary memory coherency interconnects still exist, mainly involving GPGPU vendors. AMD supports an interconnect called Infinity Architecture [3], and NVidia has NVLink [39] and NVlink-C2C [38].

The argument for developing specialized interconnects for GPUs is that they can provide higher bandwidths than what is currently available with PCIe. While these arguments may have been valid with older versions of PCIe, the latter standard has been upgraded at an unprecedented pace. PCIe Gen 7, expected to be available in 2025, will support 128MT/s per lane or 242GB/s in a ×16 card [44]. Even if proprietary interconnects were to keep improving their bandwidth, they remain proprietary. The reach of CXL reach is much wider than that of GPUs and, for that, it is a much more capable integration technology.

As with any new technology, some works have pointed to perceived CXL disadvantages, such as additional equipment cost or potential increase in software complexity [30]. For database systems, however, CXL can eliminate the memory bottleneck that has plagued them for a while—something that can be done with minimal software changes—and create possibilities for innovative architectures that were not feasible before. Any equipment cost that this might entail would be more than offset by the improving memory utilization, let alone other parallelism and acceleration opportunities that these new architectures can bring.

7 CONCLUSION

In this paper, we argued that CXL will reverse at least two decades of investments in scale-out database systems. Through seamless integration of devices' and hosts' memory, CXL allows database systems to grow via scaling up rather than scaling out, turning what is now complex distributed system development into centralized system development while simplifying the development of heterogeneous systems in the process. Given the magnitude of these possibilities, we expect CXL to catalyze the creation of a new generation of database systems with unprecedented scalability, efficiency, and ease of programming.

ACKNOWLEDGEMENTS

REFERENCES

[1] Josep Aguilar-Saborit et al. 2020. POLARIS: The Distributed SQL Engine in Azure Synapse. *Proceedings of the VLDB Endowment* 13, 12 (2020), 3204–3216. https://doi.org/10.14778/3415478.3415545

[2] Marcos K. Aguilera, Emmanuel Amaro, Nadav Amit, Erika Hunhoff, Anil Yelam, and Gerd Zellweger. 2023. Memory disaggregation: why now and what are the challenges. *ACM SIGOPS Oper. Syst. Rev.* 57, 1 (2023). https://doi.org/10.1145/3606557.3606563

[3] AMD. 2019. Infinity Architecture: The Foundation of the Modern Datacenter. https://www.amd.com/system/files/documents/LE-70001-SB-InfinityArchitecture.pdf.

[4] AsteraLabs. 2023. Industry's First CXL 2.0 RAS Capabilities Demo. https://computeexpresslink.org/resource/industrys-first-cxl-2-0-ras-capabilities-demo-with-leo-memory-connectivity-platform/.

[5] Jeff Barr. 2021. *AQUA (Advanced Query Accelerator) – A Speed Boost for Your Amazon Redshift Queries.* https://aws.amazon.com/blogs/aws/new-aqua-advanced-query-accelerator-for-amazon-redshift/

[6] Claude Barthels, Ingo Müller, Konstantin Taranov, Gustavo Alonso, and Torsten Hoefler. 2019. Strong consistency is not hard to get: Two-Phase Locking and Two-Phase Commit on Thousands of Cores. *Proceedings of the VLDB Endowment* 12, 13 (2019). https://doi.org/10.14778/3358701.3358702

[7] Carsten Binnig, Andrew Crotty, Alex Galakatos, Tim Kraska, and Erfan Zamanian. 2016. The End of Slow Networks: It's Time for a Redesign. *Proceedings of the VLDB Endowment* 9, 7 (2016). https://doi.org/10.14778/2904483.2904485

[8] Rob Blankenship and Mahesh Wagh. 2024. Introducing the CXL 3.1 Specification. https://computeexpresslink.org/wp-content/uploads/2024/03/CXL_3.1-Webinar-Presentation_Feb_2024.pdf.

[9] Andrew Chang, Jaemin Jung, Minseon Ahn, DongHun Lee, Vincent Pham, Shuyu Lyu, Jungmin Kim, and Oliver Rebholz. 2023. Expanding System Memory Boundaries Through CXL-Enabled Device-A Case Study. https://samsungmsl.com/wp-content/uploads/2023/03/MemCon2023.pdf.

[10] Monica Chiosa, Fabio Maschi, Ingo Müller, Gustavo Alonso, and Norman May. 2022. Hardware Acceleration of Compression and Encryption in SAP HANA. *Proceedings of the VLDB Endowment* 15, 12 (2022). https://doi.org/10.14778/3554821.3554822

[11] Hong-Tai Chou and David J. DeWitt. 1985. An Evaluation of Buffer Management Strategies for Relational Database Systems. In *Proceedings of 11th International Conference on Very Large Data Bases (VLDB'85).* https://doi.org/10.1007/BF01840450

[12] CXL Consortium. [n.d.]. Consortium Member List. https://www.computeexpresslink.org/members.

[13] CCIX Consortium. 2019. An Introduction to CCIX. https://www.ccixconsortium.com/wp-content/uploads/2019/11/CCIX-White-Paper-Rev111219.pdf.

[14] Bill Dally. 2011. Power, programmability, and granularity: The challenges of exascale computing. In *2011 IEEE International Test Conference (ITC'11).* IEEE Computer Society, 12–12. https://doi.org/10.1109/IPDPS.2011.420

[15] William J Dally, Yatish Turakhia, and Song Han. 2020. Domain-specific hardware accelerators. *Commun. ACM* 63, 7 (2020), 48–57. https://doi.org/10.1145/3361682

[16] Yuanwei Fang, Chen Zou, and Andrew A. Chien. 2019. Accelerating Raw Data Analysis with the ACCORDA Software and Hardware Architecture. *Proceedings of the VLDB Endowment* 12, 11 (2019). https://doi.org/10.14778/3342263.3342634

[17] Hector Garcia-Molina and Kenneth Salem. 1992. Main memory database systems: An overview. *IEEE Transactions on knowledge and data engineering* 4, 6 (1992), 509–516. https://doi.org/10.1109/69.180602

[18] Gen-Z. 2024. Gen-Z Archive. https://computeexpresslink.org/resource/gen-z-specification-archive/.

[19] Donghyun Gouk, Sangwon Lee, Miryeong Kwon, and Myoungsoo Jung. 2022. Direct Access, High-Performance Memory Disaggregation with DirectCXL. In *2022 USENIX Annual Technical Conference (USENIX ATC'22).* https://www.usenix.org/system/files/atc22-gouk.pdf

[20] Chuanxiong Guo, Haitao Wu, Zhong Deng, Gaurav Soni, Jianxi Ye, Jitu Padhye, and Marina Lipshteyn. 2016. RDMA over Commodity Ethernet at Scale. In *Proceedings of the 2016 ACM SIGCOMM Conference (SIGCOMM'16).* https://doi.org/10.1145/2934872.2934908

[21] Joseph M Hellerstein, Michael Stonebraker, James Hamilton, et al. 2007. Architecture of a database system. *Foundations and Trends® in Databases* 1, 2 (2007), 141–259. https://doi.org/10.1561/1900000002

[22] Intel. 2022. Technical Overview Of The 4th Gen Intel® Xeon® Scalable processor family. https://www.intel.com/content/www/us/en/developer/articles/technical/fourth-generation-xeon-scalable-family-overview.html.

[23] Intel. 2023. 5th Gen Intel® Xeon® Processors. https://www.intel.com/content/www/us/en/products/docs/processors/xeon/5th-gen-xeon-product-brief.html.

[24] Intel. 2024. The Intel® Xeon® 6 Processor Family. https://www.intel.com/content/www/us/en/products/details/processors/xeon/xeon6-product-brief.html.

[25] CAMEL KAIST. 2022. CAMEL's CXL Solution: A Technology Brief. https://camel.kaist.ac.kr/public/camel-cxl-memory-pooling.pdf.

[26] Dario Korolija, Dimitrios Koutsoukos, Kimberly Keeton, Konstantin Taranov, Dejan S. Milojicic, and Gustavo Alonso. 2022. Farview: Disaggregated Memory with Operator Off-loading for Database Engines. In *12th Conference on Innovative Data Systems Research (CIDR'22).* https://www.cidrdb.org/cidr2022/papers/p11-korolija.pdf

[27] Donghun Lee, Thomas Willhalm, Minseon Ahn, Suprasad Mutalik Desai, Daniel Booss, Navneet Singh, Daniel Ritter, Jungmin Kim, and Oliver Rebholz. 2023. Elastic Use of Far Memory for In-Memory Database Management Systems. In *Proceedings of the 19th International Workshop on Data Management on New Hardware (DaMoN'23).* https://doi.org/10.1145/3592980.3595311

[28] Sangjin Lee, Alberto Lerner, Philippe Bonnet, and Philippe Cudré-Mauroux. 2024. Database Kernels: Seamless Integration of Database Systems and Fast Storage via CXL. In *14th Conference on Innovative Data Systems Research (CIDR'24).* https://www.cidrdb.org/cidr2024/papers/p43-lee.pdf

[29] Alberto Lerner and Gustavo Alonso. 2024. Data Flow Architectures for Data Processing on Modern Hardware. In *40th IEEE International Conference on Data Engineering (ICDE'24).* https://exascale.info/assets/pdf/lerner2024icde.pdf

[30] Philip Levis, Kun Lin, and Amy Tai. 2023. A Case Against CXL Memory Pooling. In *Proceedings of the 22nd ACM Workshop on Hot Topics in Networks (HotNets'23).* https://doi.org/10.1145/3626111.3628195

[31] Huaicheng Li, Daniel S. Berger, Lisa Hsu, Daniel Ernst, Pantea Zardoshti, Stanko Novakovic, Monish Shah, Samir Rajadnya, Scott Lee, Ishwar Agarwal, Mark D. Hill, Marcus Fontoura, and Ricardo Bianchini. 2023. Pond: CXL-Based Memory Pooling Systems for Cloud Platforms. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'23).* https://doi.org/10.1145/3575693.3578835

[32] Clemens Lutz, Sebastian Breß, Steffen Zeuch, Tilmann Rabl, and Volker Markl. 2020. Pump Up the Volume: Processing Large Data on GPUs with Fast Interconnects. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD'20).* https://doi.org/10.1145/3318464.3389705

[33] Stefan Manegold, Peter A. Boncz, and Martin L. Kersten. 2000. Optimizing Database Architecture for the New Bottleneck: Memory Access. *The VLDB Journal* 9, 3 (dec 2000). https://doi.org/10.1007/s007780000031

[34] Hasan Al Maruf, Hao Wang, Abhishek Dhanotia, Johannes Weiner, Niket Agarwal, Pallab Bhattacharya, Chris Petersen, Mosharaf Chowdhury, Shobhit Kanaujia, and Prakash Chauhan. 2023. TPP: Transparent Page Placement for CXL-Enabled Tiered-Memory. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'23).* https://doi.org/10.1145/3582016.3582063

[35] Fabio Maschi and Gustavo Alonso. 2023. The Difficult Balance Between Modern Hardware and Conventional CPUs. In *Proceedings of the 19th International Workshop on Data Management on New Hardware (DaMoN'23).* https://doi.org/10.1145/3592980.3595314

[36] Micron. 2023. Flexible memory capacity expansion for data intensive workloads. https://www.micron.com/solutions/server/cxl.

[37] NVIDIA. 2021. NVIDIA ConnectX-7 400G Ethernet. https://www.nvidia.com/content/dam/en-zz/Solutions/networking/ethernet-adapters/connectx-7-datasheet-Final.pdf.

[38] NVIDIA. 2022. NVIDIA Opens NVLink for Custom Silicon Integration. https://nvidianews.nvidia.com/news/nvidia-opens-nvlink-for-custom-silicon-integration.

[39] NVIDIA. 2024. NVLink and NVSwitch: The building blocks of advanced multi-GPU communication—within and between servers. https://www.nvidia.com/en-us/data-center/nvlink/.

[40] OpenCAPI. 2024. OpenCAPI Specification Archive. https://computeexpresslink.org/resource/opencapi-specification-archive/.

[41] Oracle. [n.d.]. Why Oracle Exadata platforms are the best for Oracle Database. https://www.oracle.com/engineered-systems/exadata/.

[42] Oracle. 2015. SPARC S7 Processor. https://www.oracle.com/a/ocom/docs/servers/sparc/sparc-s7-processor-ds-3042417.pdf.

[43] PCI-SIG. [n.d.]. Peripheral Component Interconnect Express (PCIe) Specifications. https://pcisig.com/specifications.

[44] PCI-SIG. 2022. Announcing the PCIe 7.0 Specification. https://pcisig.com/blog/announcing-pcie%C2%AE-70-specification-doubling-data-rate-128-gts-next-generation-computing.

[45] Photowave. 2024. Optical Communications Hardware for CXL Connectivity. https://www.lightelligence.ai/index.php/product/photowave.html.

[46] Samsung. 2022. Samsung Electronics Introduces Industry's First 512GB CXL Memory Module. https://news.samsung.com/global/samsung-electronics-introduces-industrys-first-512gb-cxl-memory-module.

[47] Samsung. 2024. CXL Memory Module - Box (CMM-B). https://semiconductor.samsung.com/news-events/tech-blog/cxl-memory-module-box-cmm-b/.

[48] Samsung. 2024. Samsung CXL Solutions – CMM-H. https://semiconductor.samsung.com/news-events/tech-blog/samsung-cxl-solutions-cmm-h/.

[49] Debendra Das Sharma, Robert Blankenship, and Daniel S Berger. 2023. An Introduction to the Compute Express Link (CXL) Interconnect. *arXiv preprint arXiv:2306.11227* (2023). https://doi.org/10.48550/arXiv.2306.11227

[50] Utku Sirin, Pinar Tözün, Danica Porobic, and Anastasia Ailamaki. 2016. Micro-Architectural Analysis of In-Memory OLTP. In *Proceedings of the 2016 International Conference on Management of Data (SIGMOD'16)*. https://doi.org/10.1145/2882903.2882916

[51] Daniel J Sorin, Mark D Hill, and David A Wood. 2011. *A Primer on Memory Consistency and Cache Coherence*. Morgan & Claypool Publishers. https://doi.org/10.1007/978-3-031-01764-3

[52] Yan Sun, Yifan Yuan, Zeduo Yu, Reese Kuper, Chihun Song, Jinghan Huang, Houxiang Ji, Siddharth Agarwal, Jiaqi Lou, Ipoom Jeong, Ren Wang, Jung Ho Ahn, Tianyin Xu, and Nam Sung Kim. 2023. Demystifying CXL Memory with Genuine CXL-Ready Systems and Devices. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'23)*. https://doi.org/10.1145/3613424.3614256

[53] Yacine Taleb, Ryan Stutsman, Gabriel Antoniu, and Toni Cortes. 2018. Tailwind: Fast and Atomic RDMA-Based Replication. In *2018 USENIX Annual Technical Conference (USENIX ATC'18)*. https://www.usenix.org/system/files/conference/atc18/atc18-taleb.pdf

[54] Neil C. Thompson and Svenja Spanuth. 2021. The Decline of Computers as a General Purpose Technology. *Commun. ACM* 64, 3 (2021). https://doi.org/10.1145/3430936

[55] Alexandre Verbitski et al. 2018. Amazon Aurora: On Avoiding Distributed Consensus for I/Os, Commits, and Membership Changes. In *Proceedings of the 2018 International Conference on Management of Data (SIGMOD'18)*. https://doi.org/10.1145/3183713.3196937

[56] Alexandre Verbitski, Anurag Gupta, Debanjan Saha, Murali Brahmadesam, Kamal Gupta, Raman Mittal, Sailesh Krishnamurthy, Sandor Maurice, Tengiz Kharatishvili, and Xiaofeng Bao. 2017. Amazon Aurora: Design Considerations for High Throughput Cloud-Native Relational Databases. In *Proceedings of the 2017 ACM International Conference on Management of Data (SIGMOD'17)*. https://doi.org/10.1145/3035918.3056101

[57] Ruihong Wang, Jianguo Wang, Stratos Idreos, M. Tamer Özsu, and Walid G. Aref. 2022. The Case for Distributed Shared-Memory Databases with RDMA-Enabled Memory Disaggregation. *Proceedings of the VLDB Endowment* 16, 1 (2022). https://doi.org/10.14778/3561261.3561263

[58] XConn. 2024. World's first CXL 2.0 and PCIe Gen5 Switch IC. https://www.xconn-tech.com/product.

[59] Yuan Yuan, Rubao Lee, and Xiaodong Zhang. 2013. The Yin and Yang of Processing Data Warehousing Queries on GPU Devices. *Proceedings of the VLDB Endowment* 6, 10 (2013), 817–828. https://doi.org/10.14778/2536206.2536210

[60] Erfan Zamanian, Xiangyao Yu, Michael Stonebraker, and Tim Kraska. 2019. Rethinking Database High Availability with RDMA Networks. *Proceedings of the VLDB Endowment* 12, 11 (2019). https://doi.org/10.14778/3342263.3342639

[61] Qizhen Zhang, Philip A. Bernstein, Daniel S. Berger, and Badrish Chandramouli. 2021. Redy: Remote Dynamic Memory Cache. *Proceedings of the VLDB Endowment* 15, 4 (2021). https://doi.org/10.14778/3503585.3503587

[62] Mark Zhao, Niket Agarwal, Aarti Basant, Buğra Gedik, Satadru Pan, Mustafa Ozdal, Rakesh Komuravelli, Jerry Pan, Tianshu Bao, Haowei Lu, Sundaram Narayanan, Jack Langman, Kevin Wilfong, Harsha Rastogi, Carole-Jean Wu, Christos Kozyrakis, and Parik Pol. 2022. Understanding data storage and ingestion for large-scale deep recommendation model training: industrial product. In *Proceedings of the 49th Annual International Symposium on Computer Architecture (ISCA'22)*. https://doi.org/10.1145/3470496.3533044