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 growing trend towards specialization has led to a proliferation of accelerators and alternative processing devices. When embedded in conventional computer architectures, the PCIe link connecting the CPU to these devices becomes a bottleneck. Several proposals for alternative designs have been put forward, with these efforts having now converged into the Compute Express Link (CXL) specification. CXL is an interconnect protocol on top of PCIe with a more modern and powerful interface. While still on version 1.0 in terms of commercial availability, the potential of CXL to radically change the underlying architecture has already attracted considerable attention. This attention has been focused mainly on the possibility of using CXL to build a shared memory system among the machines in a rack. We argue, however, that such benefits are just the beginning of more significant changes that will have a major impact on database engines and data processing systems. In a nutshell, while the cloud favored *scale-out* approaches, CXL brings back *scale-up* architectures. In the paper we describe how CXL enables such architectures, and the research challenges associated with the emerging scale-up, heterogeneous hardware platforms.

Keywords: Database Engines, Memory Management, Heterogenous Computing, CXL.

1 INTRODUCTION

The cloud, demanding applications, and an ever larger amount of data are driving an increasing specialization in hardware [37]. Nowadays, many use cases rely on a wide variety of processors other than the CPU: smart NICs, FPGAs, GPUs, TPUs, DPUs, etc. In these settings, the conventional CPU-centric architecture is suboptimal. Data movement is one of the most expensive operations in a data center [11, 12], often the CPU is the least powerful element [22, 44], and the interconnect becomes a major bottleneck [19, 41].

Parallel to these developments, the cloud has become one of the bigger market drivers for computing hardware. Consequently, many current hardware developments are focused on the needs of cloud providers such as disaggregation, cost efficient use of resources (CPU, memory, storage), and the particular requirements of running virtualized environments. In such settings, the limitations mentioned above become even more acute, e.g., the bottleneck created by many VMs running on the same machine trying to access a peripheral or an accelerator (the NIC, the GPU, the local disk, etc.).

The last years have seen several competing proposals to improve both the interconnect bandwidth issue as well as the many architectural imbalances in the cloud: CCIX [9], Gen-Z [14], OpenCAPI [27], and the Compute Express Link (CXL) [31]. Today, all these specifications have been merged under the CXL umbrella, which has become the de-facto standard for future interconnects. However, CXL is much more than just an interconnect protocol that physically builds on top of PCIe. CXL supports three classes of interfaces. The *I/O interface* extends the capabilities of PCI without major changes to its semantics (essentially, copying memory regions from one device to another in a non-coherent manner). The *memory interface* allows the CPU (host) to coherently access the memory or storage of peripheral devices. And the *cache interface* allows peripheral devices to coherently access and cache data from the memory of the CPU.

The last two interfaces represent a significant change in architecture from the last decades. Cache (memory) coherency allows many agents to access memory collectively so that no agent misses each other's memory modifications. However, cache coherency methods are mostly proprietary technologies of CPU vendors and the application or the operating system cannot interact with it in any manner. Furthermore, no agents other than CPU cores can participate in the protocol, thereby establishing *coherency domains* as clear boundaries outside which memory cannot be accessed collectively.

This is one of the big architectural barriers that CXL removes, enabling what are called *Type 2* devices (e.g., CPUs, GPUs, or FPGAs) to directly access the host and each other's memory in a coherent fashion. Big as this step is, it not the only one. With version 3.0, CXL goes well beyond the traditional role of an interconnect and becomes a rack level networking fabric that is both more performant than current Ethernet based systems (CXL 3.0 claims 64 GBytes/s bandwidth versus 100 or 200 Gbits/s in data centers) as well as more powerful in terms of its interface (coherent access, memory sharing, encryption, etc., versus simple reliable packet sending and receiving in TCP/IP or data copying in RDMA).

In this paper we focus on this capability of CXL and discuss how it enables *scale-up* (or vertically scaled) database engines that will significantly differ from the *scale-out* (horizontally scaled) systems that have dominated the landscape in the last years due to the constraints imposed by cloud architectures. We first provide a brief introduction to CXL and then discuss in detail new architectural possibilities enabled by novel CXL features as well as the many challenges that will have to be addressed before they can be exploited to full advantage.

Preprint, under review.

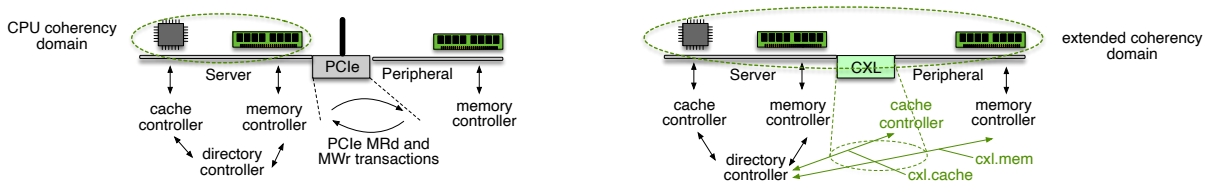

Figure 1: (left) Peripherals connected using PCIe cards are outside the coherency domain even if they contain memory. (right) CXL extends the coherency domain by allowing the peripheral to communicate with the directory controller.

2 BACKGROUND AND MOTIVATION

In this section, we introduce CXL and its basic mechanisms and discuss the fundamental changes these mechanisms allow.

2.1 CXL Status

CXL is a public consortium lead by Intel. It has been evolving quickly since its inception in 2019, and so far, there have been four major spec releases: 1.1, 2.0, 3.0, and 3.1. The 1.1 version is centered around local memory expansion, i.e., allowing a server to access more memory than available in its DIMM slots. Release 2.0 introduced basic forms of CXL interconnects (e.g., switches), so that memory in a remote chassis is accessible, too, and support for memory pooling. A server can aggregate memory from several expanders and vice-versa. Releases 3.0 and 3.1 added support for more sophisticated networking and for sharing memory across multiple servers and peripherals, rather than the latter expanding memory of the former.

At the time of writing, the CXL hardware available is overwhelmingly based on version 1.1, although some vendors have already adopted selected features of 2.0 (in what some refer to informally as 1.1+). Versions 3.0 and 3.1 require improvements in PCIe, which will be accomplished in the already ratified Gen 6 of that protocol. Presently, one can hardly avoid CXL since all major `x86` server class CPU vendors have adopted it.

2.2 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. Naturally, issues of data consistency may arise. *Cache coherency* 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 [34]. This means that if a core wishes to modify a cached line, all other copies residing in different cores must be invalidated first. Therefore, the system keeps track of the state in which each cached line is. Read-only copies are deemed *Shared* (S), while invalidated copies are called *Invalid* (I). A unique copy is called *Exclusive* (E) while it is still intact, or *Modified* (M) after its contents have been updated w.r.t. main memory. Other states may exist, but these four are the most common. Often, coherency protocols announce the states they use. What we described above is a `MESI` protocol, which is what CXL uses.

In current computers, only caching done by CPU cores is tracked. We say that the *coherency domain* extends only to the CPU. Data copies outside this domain are not coherent in the sense that the hardware will not keep track of whether that copy is exclusive. Therefore, a conflict may arise if one updates it.

2.3 Coherent and Non-Coherent Data Transfers

Non-coherent transfers occur between servers and peripherals through PCIe exchanges called *transactions*. A transaction is a sequence of messages in the context of a read or a write operation between a server's and a peripheral's memory. Figure 1 (left) shows how this exchange is done vie PCIe. Note that the communication takes place directly between memory controllers.

In contrast, coherent transfers between CPU cores and memory are not direct. They involve additional components that track the information necessary to maintain coherency. To `load` a line, a CPU core uses a local Cache Controller (CC). In turn, the CC notifies a (socket central) component, called a Directory Controller (DC), of the intent to access that line. The DC keeps track of all lines currently cached and can evaluate if, to maintain the invariants, this new request should trigger any cache invalidation. Only after the invalidations are performed can the request be forwarded to the memory controller. Figure 1 (left) also depicts this process, showing that the peripherals are not on the coherency domain.

CXL extends the type of transactions that can occur between servers and peripherals. It is backward compatible with PCIe transactions, which it wraps under a sub-protocol called `cxl.io`, and it adds two other sub-protocols, `cxl.mem` and `cxl.cache`. The server uses the former to communicate with the peripheral's memory controller as if it resided locally on its motherboard. The peripheral uses the latter sub-protocol to cache contents managed by the server's directory controller. Through these two sub-protocols, any memory the device offers can be seamlessly incorporated into the overall system, and the device can cache data that lives in the system's memory. Figure 1 (right) depicts this scenario. Due to space constraints, this paper will focus on devices that only use `cxl.io` and `cxl.mem`, called (*Type 3*) devices. Devices such as CPUs, GPUs, or FPGAs (*Type 2*) use both `cxl.mem` and `cxl.cache`, and some devices use `cxl.cache` only (*Type 1*).

2.4 CXL Performance Characterization

Current conventional NUMA servers typically comprise two sockets, each containing a CPU and half of the system's DRAM. Such tight coupling of CPU and memory is problematic as it prevents the two sides to be scaled independently. CXL can be used to add coherent memory without having to increase the number of sockets in the machine. Storage vendors such as Micron and Samsung are launching a new device category called *Memory Expander* [23, 30]. A memory expander allows a server to map more memory than the DRAM DIMMs (memory slots) it carries on its motherboard by simply plugging in what looks like a special type of SSD device that carries DRAM memory instead. Memory expanders not only

increase DRAM size, they also improve memory bandwidth because they effectively add memory controllers to the system.

Preliminary benchmarks that use actual CXL implementations are already available [35]. Regarding latency, executing a `load` instruction against a given type of CXL-attached memory can be 35% longer than the equivalent NUMA memory access. Executing a `store` under the same conditions can present slightly lower but equivalent overheads. Regarding bandwidth, it helps to measure the efficiency of the transfer, i.e., what percentage of the nominal memory bandwidth capacity can be achieved. Transfers from NUMA nodes can be 70% efficient when considering only `loads`, compared to 46% efficiency when reading the same type of memory through a CXL interconnect. Curiously, `stores` can be more bandwidth efficient if directed to a CXL device than a neighbor NUMA socket, 12% higher. The reason is that `stores` to a CXL device can bypass several coherency checks that must occur on a NUMA node.

These micro-benchmark results have been complemented with more end-to-end studies. In a recent article by Meta [21], CXL memory is used to store cold pages with the operating system swapping pages back and forth between the host DRAM and the CXL memory. In essence, CXL memory is used as an additional memory tier between the DRAM and storage. The results suggest that the bandwidth available from CXL memory will be around 64 GB/s with latency only slightly larger than that of NUMA memory. Similarly, a study from Microsoft analyzed the impact of CXL memory for their cloud environment [18]. While the results differ from workload to workload, the study found that under the expected latency increases, some 26% of the 158 workloads studied show less than 1% performance penalty due to it, and an additional 17% show less than 5%. In contrast, 21% of the workloads were affected by more than 25% of performance decrease. For database workloads, specifically TPC-H, the overheads are highly query-dependent but are 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

The studies above suggest another disruptive change that CXL can bring about [18, 21, 35]. CXL interconnects—fabrics carrying coherency traffic across servers—are expected to be significantly more efficient than traditional networks, even RDMA based ones. The reason is that RDMA requires conversions between the Infiniband/RoCE protocols and the PCIe protocol. In other words, a NIC uses PCIe to talk to its host but Infiniband/RoCE is use to interact with the rest of the system. In contrast, all traffic in CXL is already carried by PCIe-compatible messages.

Some studies have attempted to quantify these benefits [15]. The latency of CXL communications was found to be in the three hundred nanoseconds range, while the fastest exchanges in RDMA take at least 2.7 microseconds—a difference of 8.3×. The study also hints at a structural advantages that gives CXL more bandwidth than traditional networking: NICs are sub-dimensioned w.r.t. the number of PCIe lanes they occupy. For instance, a 400 Gbps NIC (50 GB/s) will occupy 16 PCIe Gen5 lanes that, in the aggregate, can offer 64 GB/s [24]. Put differently, over 20% of the available PCIe bandwidth does not translate into network bandwidth. This discrepancy has been consistent through different network speeds and will likely remain the same for 800 Gbps and 1.6 Tbps NICs when PCIe Gen 6 and 7 servers are available.

We note that with CXL versions 3.0 and 3.1 the networking works at a peer-to-peer level as well. Two peer peripherals can access each other's memory independently of the server. Most importantly, CXL 3.1 brings the concept of *Global Integrated Memory*. This feature allows several servers and peripherals to contribute a region of memory mapped globally, i.e., many servers and peripherals share the same area. For those reasons, CXL is poised to become a much better way to connect CPUs, memory, and storage, at least on a rack level. CXL strength is allowing coherency domains that encompass full racks.

3 SHARED MEMORY ARCHITECTURES

In this section we discuss how CXL memory could change the architecture of database engines and the advantages it would bring.

3.1 Single Machine Memory Expansion

Databases maintain a pool of shared memory available to the threads where queries or transactions run. The so called *Buffer Cache* or *Buffer Pool* is used as an intermediate stage and cache between persistent storage and the private space used by every processing thread. In the Buffer Pool the data is stored and read by the processing threads, with the engine implementing diverse cache replacement policies to minimize the number of page misses and the traffic to storage. This architecture makes sense in an engine that runs on a single machine. However, it is not without problems: the working space of every thread, the storage needed for indexes and auxiliary data structures, and the memory used by the engine all compete for space with the Buffer Pool.

A simple way to exploit CXL memory follows the tiered memory approach pursued by Meta and mentioned above [21]. The configuration used is shown in Figure 2(a) where CXL exposes the memory from a peripheral device (in this case a memory device) and seamlessly integrates it with the server's memory. In such a set up, the CXL memory is logically allocated above the local DRAM memory and pages are swapped back and forth as needed. The idea is the same as with a disk but using DRAM and faster interconnect which provides much lower latencies. In a database engine, the paging system could be adapted to the needs of data processing instead or relying on general purpose page replacement policies [8]. In general, CXL used in this way brings the benefits and challenges of a deeper memory hierarchy and existing techniques should be revisited accordingly [20, 33].

While a deeper memory hierarchy would facilitate adoption, CXL memory does not need to be treated as block device. More interesting configurations arise when the local DRAM is used to, e.g., store indexes for fast traversal with the data actually in the CXL memory where much more capacity can be provided and the data can be accessed on a tuple basis rather than as a block.

Another important aspect that is often overlooked is that CXL memory frees up the system from inter-dependencies. In existing CPUs, the type of DIMM supported is tied to the CPU architecture. Also, the socket organization of CPUs requires memory capacities proportional to the number of sockets and cores. CXL memory removes all these constraints. In terms of capacity, the memory on

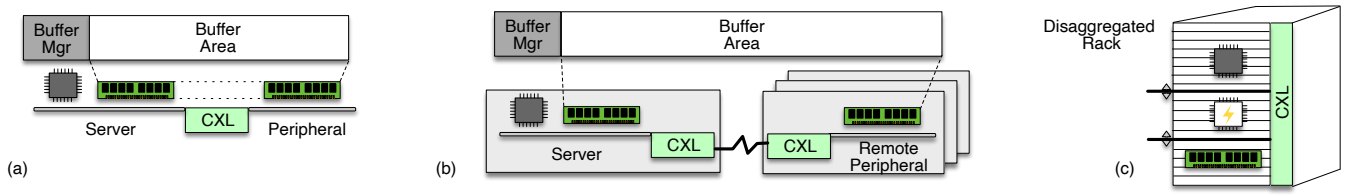

Figure 2: CXL allows progressive levels of desegregation: (a) local and (b) far memory expansion, and (c) full-rack disaggregation.

CXL is not tied to the CPU architecture so one could add DIMMs different from those used in the CPU. It can also be used to simply add more memory capacity independently of the number of cores. Doing so offers more flexible options to configure the database by varying the proportion of cores to memory.

Finally, main memory databases and hybrid transactional/analytic systems could greatly benefit from the architecture. An interesting configuration to explore would be to place the transactional workload on the local DRAM and use CXL memory for the analytic part, also providing space for more data as well as structures like data cubes, materialized tables, de-normalized tables, etc., without that affecting the transactional workload in any way.

From these ideas, several research directions open up:

- How should the memory extension be handled? As a block device with pagination or as addressable memory just like DRAM on a CPU socket? The former makes integration easier but the latter is likely to being more performance and design opportunities.
- Is the memory expansion fast enough for OLTP or will be suitable mainly for OLAP? Can it be used to perform both on the same machine and what are the implications?
- What data structures should be kept in the local memory and which ones on the memory expansion? Are these data structures suitable for the increases non-uniformity in memory accesses that CXL memory creates?

3.2 Disaggregated Memory

The limitations imposed by the memory available in a given machine have been recognized long ago. In the cloud, the problem becomes worse due to what has been called *stranded memory*: the fact that, in the cloud, a machine often runs out of virtual CPUs to rent before it runs out memory, resulting in parts of its DRAM being unused. Since memory is one of the most expensive components in today's data centers, this is a major source of inefficiencies [18].

In the context of database engines, how much main memory to allocate to the engine is crucial for performance purposes. In the cloud this is made worse by the fact that storage is disaggregated and swapping of pages involves networking. To minimize this effect, many cloud providers support special configurations for databases using network attached block storage and disks that are much faster than those typically used for conventional applications—of course, with the corresponding increase in price as these services are costlier than typical cloud systems. The network overhead and the slow storage needed (for economic reasons) when the data is very large has led over time to additional layers in the system such as main memory caches and even specialized solutions with SSDs and FPGAs directly attached to the network [4].

Recently, researchers have starting exploring ways to provide additional DRAM memory to a sever without necessarily attaching it to this server also in the context of databases [2, 16, 40, 43]. The ideas has led to the notions of *remote memory*, *far memory*, or *disaggregated memory*. This is implemented in different ways. Far memory is a term typically used to refer to any generic configuration where the additional DRAM is not local. Remote memory generally refers to utilizing unused memory of other machines. Disaggregated memory is memory not allocated to any machine but available for servers to use. In most cases, the data is exchanged 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.

As Figure 2(b) shows, a CXL peripheral can be remote, i.e., it can sit outside the server's chassis and use the CXL interconnect for integration. This feature allows a server to pool memory from several CXL devices and, conversely, for a remote device to carve its memory across different servers. In this way, CXL enables disaggregated memory with the advantage of the memory being coherent which is not the case when disaggregated memory is implemented through the network. Through the addition of multi-level switches to CXL, 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. Such an approach allows to create a large scale memory pool (with coherency enforced by hardware) that is far more efficient that what can be accomplished in a distributed system.

An intriguing use of such a configuration is database migration and elasticity across machines. Databases have a large state and are not very elastic as they are heavily anchored by the data loaded in memory so that the system is reasonably fast. Disaggregated memory implemented through CXL would allow to place the Buffer Pool on the disaggregated memory and use the local DRAM just 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 and is a good trade-off in return for far more elasticity than it 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, interesting questions arise:

- Implementing these features will require rethinking the internal database architecture to remove the assumption that everything

is in local memory. What needs to be local and what can be placed in disaggregated memory will require extensive experimentation to determine which part of the engine can tolerate the additional latency of CXL memory.
- With the suitable architectural approach, engines can become far more elastic as pointed out above. Should this be done at the level of entire engines 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 a dynamically changing multiprogramming level?

3.3 Shared Memory = Scale Up

Distributed databases are a typical way to implement larger systems [1, 28, 39]. However, the architecture of the engines discussed above does not scale as expected: cache invalidation now crosses machines; updates imply distributed, more onerous locks; there is wasted memory as the same data is copied into the local buffer cache of several machines, etc. These problems are addressed through a mix of replication and sharding, increasingly using RDMA to reduce the network overhead [5, 6, 36, 42]. Invariably a fragile balance is reached 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., [38]).

CXL supports building a larger database in a truly scaled-up manner rather than through distribution. This is thanks to CXL's ability to integrate devices via a coherency domain that can encompass the entire rack. In particular, CXL supports *Global Integrated Memory* (GIM), where each system component contributes a range of its own memory to the collective memory, rather than having each machine expand its memory by monopolizing it out of a pool. Since now each server plus the memory expanders can share memory, the boundaries of the machines in a rack get blurred. The entire rack can be seen as a single machine. Figure 2(c) depicts this scenario.

Rack-level integration is arguably the most impactful change CXL enables. It liberates the database system from managing consistency across servers by moving the memory unification effort to hardware. Moreover, as discussed above, CXL offers a bandwidth far larger than what is available with today's networks and with far lower latency. The result is a "blank canvas" for database architects with almost none of the disadvantages of the previous scalability methods. We expect this to cause a radical change in the way scalable databases and data processing engines are designed.

Promising as the prospects are, the road ahead is not without challenges. The last two decades have veered away from scale-up systems, focusing instead on cloud-native and scale-out approaches. Moreover, many existing large systems carry biases that do not exist anymore, e.g., that networks are slow or that I/O is prohibitively expensive. A fully disaggregated system like the one CXL enables breaks new ground regarding the algorithms and data structures. Here are some of the questions it opens:
- Since our fundamental data operations are built around hashing and sorting, do we know how to conduct these operations on a rack-level scale? Do we fully understand when to use which?
- Given that each core now can 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? In particular, how is the *coherency traffic* generated by a typical data structure? Given that the invalidation messages can dominate access time, can it be improved?
- 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?

The benefits of answering these questions are significant. For instance, consider transactional workloads. By keeping the data in one (large) location, transactional updates can be done against a centralized buffer cache, rather than across a distributed system. Similarly, many of the data structures the database has to maintain can now be centralized in the disaggregated memory instead needing to be synchronized across the machines involved.

4 NEAR-DATA PROCESSING

CXL indirectly enables another set of important features beyond integrating local, far, and disaggregated DRAM. To understand these features, it helps to look into how a CXL device is implemented—and a comparison to Persistent Memory (PMem) devices is relevant here. Persistent Memory solutions like Intel's Optane carry a complex controller that is embedded into the PMem DIMM. In contrast, CXL support involves wrapping existing DRAM DIMMs (or other types of memory) within a specialized hardware memory controller (cf. Figure 1 (right)) using a variety of possible PCIe device form factors. The device's controller, which is transparent to the application, must implement the CXL protocol and potentially manage several logical devices carved out of the physical one, to cite one functionality. The controller will likely utilize a processor that implements the protocol, performs traffic management, and arbitrates the devices' requests. The CXL memory controller is likely to be rather sophisticated, probably in the same way that SSD controllers are far more than simple data relays to flash storage.

We argue that a sophisticated controller, potentially using a specialized processor, opens a tremendous opportunity: part of the CXL controller's computing power can be co-opted to perform near-data processing. Near-data processing is known to optimize away unnecessary data movements, making the entire architecture far more efficient than one that simply integrates memory. Putting it differently, a CXL controller can functionally behave as a smart NIC, following the idea that CXL could replace the network. However, unlike smart NICs, the CXL controller will be sitting not only next to compute devices (CPUs, FPGAs, GPUs) but also close to memory and storage. Figure 3 (top) depicts this scenario.

This idea is very much aligned with current trends. In research there have been efforts to, e.g., explore the advantages for databases of placing a small processor near memory [13] or in disaggregated memory [16]. Oracle had a processor (SPARC S7) with *Data Analytics Accelerators* (DAX) placed between the cores and the memory where basic relational operators could be offloaded to filter data before it hits the processor caches [29]. Recently, Amazon developed AQUA, a network attached SSD caching layer for Redshift that used an FPGA to offload relational operators to the caching layer [4].

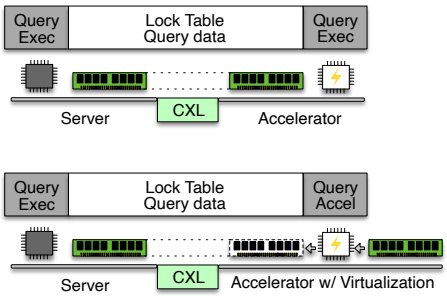

Figure 3: Near-data processing under CXL. (top) A processor or FPGA managing the expanded memory can be co-opted to execute a portion of a query. (bottom) A unique opportunity for acceleration exists through virtual memory regions.

CXL offers a way to better implement these ideas but not without challenges, some of which we list below:

- How to perform query processing near the data? This possibility is not new, as several query operators have been shown to have near-data implementations. Compression and decompression, encryption and decryption, selection, projection, filtering with LIKE predicates, and a wide range of other relational operators that have been demonstrated to bring substantial advantages in practice [4, 7, 13, 16].
- Do we leverage the controller to implement the functionality or do we attach an accelerator in the CXL path? How does such a controller or such an accelerator look like?
- If the lock table is also placed in the shared memory region, then even updates to common data structures from both sides of the query could be performed. What are suitable data structures that can be manipulated by both "sides?" In other words, are the invalidation traffic of traditional data structures acceptable, if they are updated by several processors?
- Lastly, a very intriguing idea is to use the CXL controller to implement a *virtual memory region* that does not correspond to actual content but to a *service* that takes data from the memory and transforms it before sending it to the requester to make it look like it was stored in memory all along. Figure 3 (bottom) illustrates such a use case. In databases this can be used to implement, e.g., view materialization on-the-fly; data type transformations; data transpositions from column to row, row to column, or matrix transpositions [16]; on-the-fly data cubes and statistical summaries; or even to better integrate databases and smart storage systems [17].

5 HETEROGENEOUS ARCHITECTURES

GPUs, TPUs, DPUs, smart NICs, and FPGAs configured to perform different types of near-data processing or processing in memory acceleration have been shown to be advantageous in several real systems scenarios. If anything, this trend has exposed many of the limitations of CPU-centric computer architectures and, as discussed, the limitations of existing interconnects. With around 64 GB/s bandwidth per slot and a typical server with 4 to 6 slots, CXL enables the creation of a computing platform much more flexible than a rack of servers with shared memory. A federation of heterogeneous processing nodes can now operate on a unique coherency domain.

This composability possibility that CXL creates opens a research field of its own: if computational devices are independently connected, what should a heterogeneous machine look like? Can we (and should we) build machines that accommodate specific workloads better than others? 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. With a heterogeneous architecture that seamlessly integrates CPUs and GPUs, it becomes possible to implement ML operators directly on the database engine while still taking advantage of suitable hardware. This will require changes to the engine design and architecture. However, given the powerful compute fabric that CXL enables, it will likely lead to a new generation of scale-up database engines.

6 RELATED EFFORTS

CXL is, in no small part, the result of consolidating several projects that came before it, most notably CCIX [9], GenZ [14], and OpenCAPI [27]. An overwhelming number of institutions and companies are working together to advance the protocol [10]. 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 across its GPGPUs [3]. NVidia has alternative interconnect technology in the form of NVLink [25] and NVlink-C2C [26].

The argument for developing these highly specialized interconnects, especially for GPGPUs, is that they can provide (a) a much higher bandwidth than what is possible with PCIe or the network and (b) a unified memory between the host and the accelerator to be able to exchange pointers and to avoid having to pin memory when exchanging data. While these arguments may have been valid in 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, i.e., 242GB/s in a ×16 card [32]. Even if the proprietary interconnect improved their bandwidth, they would still carry the usual drawbacks: outside of the few CPU/GPGPUs they support, they bring inefficient memory use, data copying overheads, consistency issues, etc. It remains to be seen how these competing standards will evolve, merge, or co-exists.

7 CONCLUSION

In this paper, we argued that the availability of CXL technology will upend at least two decades of investments in scale-out database systems. The reason is that, through a quite natural memory integration, CXL allows database systems to scale up instead. The appeal of this direction change is that it turns what is now complex distributed system development into familiar centralized system development instead. We discussed how CXL supports a range of disaggregated—but coherent!—memory settings in a system, from local- to far-memory expansion to full rack-level disaggregation. At each of these steps, we presented several research questions these changes open, along with the benefits of addressing them. Lastly, we argued that CXL can also support near-data processing and heterogeneous platforms in uncomplicated ways never available before. Given the magnitude of these possibilities, we expect CXL to foster the design of an entirely new generation of database systems with unprecedented scalability, efficiency, and integration.

REFERENCES

[1] Josep Aguilar-Saborit et al. 2020. POLARIS: The Distributed SQL Engine in Azure Synapse. *Proc. VLDB Endow.* 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. [n.d.]. Infinity Architecture: A New Era in Accelerated System Connectivity. https://www.amd.com/en/technologies/infinity-architecture.

[4] 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/

[5] 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. *Proc. VLDB Endow.* 12, 13 (2019). https://doi.org/10.14778/3358701.3358702

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

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

[8] Hong-Tai Chou and David J. DeWitt. 1985. An Evaluation of Buffer Management Strategies for Relational Database Systems. In *VLDB'85, Proceedings of 11th International Conference on Very Large Data Bases, August 21-23, 1985, Stockholm, Sweden.* https://doi.org/10.1007/BF01840450

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

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

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

[12] 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

[13] 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

[14] GenZ. [n.d.]. GenZ Archive. https://www.computeexpresslink.org/projects-3.

[15] Donghyun Gouk, Sangwon Lee, Miryeong Kwon, and Myoungsoo Jung. 2022. Direct Access, High-Performance Memory Disaggregation with DirectCXL *(USENIX ATC'22').* https://www.usenix.org/conference/atc22/presentation/gouk

[16] 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 2022, Chaminade, CA, USA, January 9-12, 2022.* https://www.cidrdb.org/cidr2022/papers/p11-korolija.pdf

[17] 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 2024, Chaminade, CA, USA, January 9-12, 2022.* http://exascale.info/assets/pdf/lee2024cidr.pdf

[18] 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 *(ASPLOS 2023).* https://doi.org/10.1145/3575693.3578835

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

[20] 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

[21] 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 *(ASPLOS 2023).* https://doi.org/10.1145/3582016.3582063

[22] Fabio Maschi and Gustavo Alonso. 2023. The Difficult Balance Between Modern Hardware and Conventional CPUs *(DaMoN'23).* https://doi.org/10.1145/3592980.3595314

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

[24] NVidia. [n.d.]. NVidia ConnectX-7 400G Ethernet. https://www.nvidia.com/content/dam/en-zz/Solutions/networking/ethernet-adapters/connectx-7-datasheet-Final.pdf.

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

[26] NVIDIAG. [n.d.]. NVIDIA Opens NVLink for Custom Silicon Integration. https://nvidianews.nvidia.com/news/nvidia-opens-nvlink-for-custom-silicon-integration.

[27] OpenCAPI. [n.d.]. OpenCAPI Archive. https://www.computeexpresslink.org/occ-archive.

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

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

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

[31] Debendra Das Sharma. [n.d.]. Compute Express Link. https://docs.wixstatic.com/ugd/0c1418_d9878707bbb7427786b70c3c91d5fbd1.pdf.

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

[33] Utku Sirin, Pinar Tözün, Danica Porobic, and Anastasia Ailamaki. 2016. Micro-Architectural Analysis of In-Memory OLTP *(SIGMOD '16).* https://doi.org/10.1145/2882903.2882916

[34] 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

[35] 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 *(MICRO'23).* https://doi.org/10.1145/3613424.3614256

[36] Yacine Taleb, Ryan Stutsman, Gabriel Antoniu, and Toni Cortes. 2018. Tailwind: Fast and Atomic RDMA-Based Replication *(USENIX ATC'18).* https://www.usenix.org/conference/atc18/presentation/taleb

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

[38] Alexandre Verbitski et al. 2018. Amazon Aurora: On Avoiding Distributed Consensus for I/Os, Commits, and Membership Changes *(SIGMOD '18).* https://doi.org/10.1145/3183713.3196937

[39] 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 *(SIGMOD'17).* https://doi.org/10.1145/3035918.3056101

[40] 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. *Proc. VLDB Endow.* 16, 1 (2022). https://doi.org/10.14778/3561261.3561263

[41] 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

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

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

[44] Mark Zhao et al. 2022. Understanding data storage and ingestion for large-scale deep recommendation model training: industrial product *(ISCA'22).* https://doi.org/10.1145/3470496.3533044