Near to Far: An Evaluation of Disaggregated Memory for In-Memory Data Processing

Andreas Geyer
andreas.geyer@tu-dresden.de
Technische Universität Dresden
Dresden, Germany

Johannes Pietrzyk
johannes.pietrzyk@tu-dresden.de
Technische Universität Dresden
Dresden, Germany

Alexander Krause
alexander.krause@tu-dresden.de
Technische Universität Dresden
Dresden, Germany

Dirk Habich
dirk.habich@tu-dresden.de
Technische Universität Dresden
Dresden, Germany

Wolfgang Lehner
wolfgang.lehner@tu-dresden.de
Technische Universität Dresden
Dresden, Germany

Christian Färber
christian.faerber@intel.com
Intel Deutschland GmbH
Feldkirchen, Germany

Thomas Willhalm
thomas.willhalm@intel.com
Intel Deutschland GmbH
Feldkirchen, Germany

Abstract

Efficient in-memory data processing relies on the availability of sufficient resources, be it CPU time or available main memory. Traditional approaches are coping with resource limitations by either adding more processors or RAM sticks to a single server (scale-up) or by adding multiple servers to a network cluster (scale-out). Further, the InfiniBand interconnect enables Remote Direct Memory Access (RDMA) and thus enhances the possibilities of resource sharing between distinct servers. Resource disaggregation means the (dynamic) sharing of available hardware, e. g., through the network. This paradigm is now further enhanced by the specification of Compute Express Link (CXL). In this paper, we systematically evaluate the implications of memory expansion as a form of resource disaggregation from the perspective of in-memory data processing through the local Ultrapath Interconnect (UPI), RDMA via InfiniBand, and PCIe attached memory via CXL. Our results show that CXL yields behavior that is comparable to UPI and outperforms the inherently asynchronous RDMA connection. Further, we found that handling UPI-attached memory as a type of disaggregated resource can yield additional performance benefits.

DIMES '23, October 23, 2023, Koblenz, Germany

ACM ISBN 979-8-4007-0300-3/23/10…$15.00
https://doi.org/10.1145/3609308.3625271

CCS Concepts: • **Information systems → Main memory engines**; *Online analytical processing engines*; • **Hardware → Emerging interfaces**.

Keywords: Memory Disaggregation, CXL, RDMA, UPI

ACM Reference Format:
Andreas Geyer, Johannes Pietrzyk, Alexander Krause, Dirk Habich, Wolfgang Lehner, Christian Färber, and Thomas Willhalm. 2023. Near to Far: An Evaluation of Disaggregated Memory for In-Memory Data Processing. In *1st Workshop on Disruptive Memory Systems (DIMES '23), October 23, 2023, Koblenz, Germany*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3609308.3625271

1 Introduction

The continuous evolution of hardware is an inherent characteristic of this technology. Up to now, main memory and CPU have been the most decisive drivers, but now other components like networking and respecting protocols have become innovation drivers as well. Today, we already see several hardware architectures, e. g., scale-up or scale-out solutions to cope with the ever increasing demand for more computational power and storage space. Over time, multi-socket systems have emerged within the scale-up architecture and introduced the now well-known Non-Uniform Memory Access (NUMA) effect [8, 9, 13]. However, still all communication happens on the same machine and is – depending on the interconnect and support by specific components [22] – reasonably fast.

The dawn of InfiniBand as a high speed interconnect enabled, e. g., cloud providers to more efficiently share potentially unused resources between individual machines. In that field, resource disaggregation is becoming increasingly prevalent. That is, instead of hosting multiple servers, providers now stack up on dedicated resource cards, i. e., racks full of network-attached circuit boards holding either

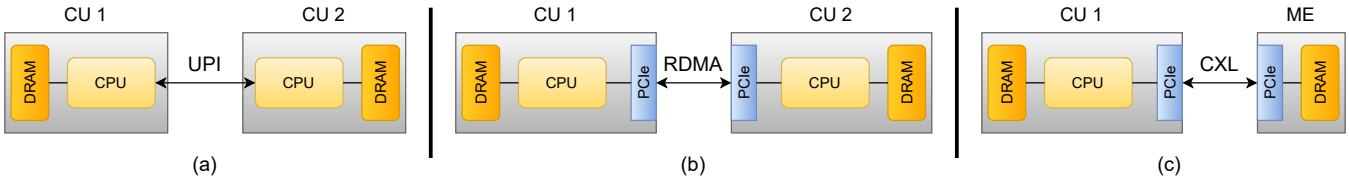


Figure 1. Schematic overview of different memory expansion methods using distinct Compute Units and a Memory Expansion.

CPUs, DRAM, NVMe drives, accelerators like GPUs or FPGAs, and so on. Currently, disaggregation is realized through Remote Direct Memory Access (RDMA), where two or more servers are connected by, e. g., an InfiniBand cable and share their main memory with each other. Yet, the next big player is just around the corner: Compute Express Link (CXL) [17]. This new PCIe based communication standard was first described in 2019, was later fused with Gen-Z [18] and is now just arriving at the market. CXL features multiple protocols, such as `CXL.io`, `CXL.cache` and `CXL.mem`, to provide, e. g., cache-coherent memory access for a remotely attached memory block. Through CXL, we are now able to combine multiple resources through the network to a single *software composed system*. However, this new architecture comes with well-known problems: Physically distributed memory leads to even more severe NUMA effects. That is because CXL-attached memory extends the coherent virtual address space by being exposed as just another NUMA node.

Contribution and Outline. In this paper, we systematically investigate the implications of resource disaggregation through both RDMA and CXL from the perspective of in-memory data processing. The most intriguing questions are: (i) Can traditional solutions, that worked for scale-up servers, be applied to disaggregated systems as well? (ii) What are key similarities or differences? To answer these questions, we first conduct several microbenchmarks of representative in-memory processing operators from modern database engines [4, 14] and thus, Section 2 will introduce our experimental methodology. Sections 3 and 4 show our findings, clustered by operator complexity. Then, we extend our evaluation focus to a recently proposed optimization technique in Section 5. After this evaluation, we give an overview about related work in Section 6. Finally, this paper closes with a summary and a discussion of future work in Section 7.

2 Evaluation Methodology

Main memory is an important resource for data processing and its availability can be increased through several methods. In this paper, we systematically investigate memory expansion through (i) scale-up with UPI, (ii) scale-out-like with RDMA over InfiniBand, and (iii) resource disaggregation with a CXL attached DRAM card. To study the architectural implications, we developed a comprehensive evaluation prototype in C++ according to Figure 1. Further, we designed different benchmarks, based on representative in-memory data processing operators from database engines with both a trivial (e. g., aggregation) and complex (e. g., hash join) internal state to showcase, how memory distances and synchronous vs. asynchronous access influence the processing behavior. Currently, we do not have access to a single setup that has both CXL and RDMA hardware. Thus, UPI and CXL benchmarks, cf. Figure 1(a,c), are executed on a different physical setup than the RDMA benchmarks, cf. Figure 1b. For every benchmark, we refer to *local* and *remote* data, where *local* refers to data that is located on the same NUMA node as the processing CPU core.

Hardware Setup. The benchmarks for Figure 1(a,c) are executed on a single server with 4th-generation Intel® Xeon® Scalable processors (code-named "Sapphire Rapids"), i. e., two Intel® Xeon® Platinum 8480+ processors, with each having 56 physical cores with a base frequency of 2.0 GHz. Main memory consists of 16 DDR5 memory DIMMs with 16 GB each, which results in 128 GB of memory per processor. Since the processors do not support CXL type 3 devices, the Host to Memory Expansion (ME) interface uses a CXL Type 2 link (not to be confused with CXL 2.0, cf. [1]) but utilizes `CXL.mem` transactions. The experimental ME is realized using an Intel® Agilex™ I-Series FPGA Development Kit (DK-DEV-AGI027R1BES), which supports PCIe 5.0 x16 CXL 1.1 connectivity and DDR4 memory. This card hosts 2x 8 GB SR DDR4-2666 components soldered to it and is hence exposing 16 GB of memory as a third NUMA-node to the system.

RDMA benchmarks are executed on a second server setup consisting of two machines. One server is equipped with four Intel® Xeon® Gold 6130 with 16 cores each and a base clock of 2.1 GHz, the second server features four Intel® Xeon® Gold 5120 with 14 cores each and a base clock of 2.2 GHz. On both machines, each CPU is directly connected to 96 GB of local memory. A Mellanox ConnectX-4 card with up to 100 GBit/s (4xEDR) is connecting the NUMA socket 0 of both machines. Therefore, we pin working threads and memory usage solely to NUMA-node 0 on both machines when conducting the RDMA experiments.

Ultrapath Interconnect Setup. UPI is an interconnect between processors inside a NUMA machine, i. e., it connects NUMA nodes and allows for cache-coherent data transfer inside the coherent memory space. Figure 1a illustrates the general setup. For these experiments, local data is placed on NUMA socket 0 and remote data is placed on another node, which is one NUMA-hop away, but still inside the very same

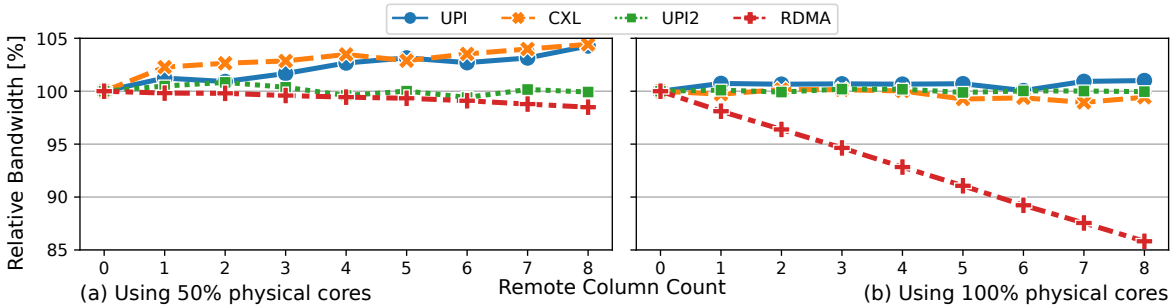


Figure 2. Relative behavior of UPI, CXL and RDMA with additional remote buffers for the aggregate kernel. Threads for remote buffers are first assigned to physical cores if available, otherwise to hyperthreads.

physical machine. This placement requires the processing thread to access the data synchronously through UPI and to transfer it into the local processor cache to process it.

Remote Direct Memory Access Setup. Figure 1b illustrates the connection of two servers via PCIe-attached InfiniBand (IB) adapters. In this setup, remote data is placed on a physically different machine than the local data. The PCIe slot is directly attached to the NUMA socket, where the local data is stored and accessing the remote data is performed asynchronously through the IB verbs read/write.

Compute Express Link Setup. Our CXL setup involves the experimental hardware prototype, which allows us to emulate the basic CXL behavior. Local data is stored on the host and remote data is placed on NUMA node 3, which is directly connected to both NUMA sockets of the host system. Allocating memory on this specific node is performed through the `libnuma` call `numa_alloc_on_node()` and data access is as simple as dereferencing a pointer. The compute capabilities of the FPGA are not used in the conducted experiments.

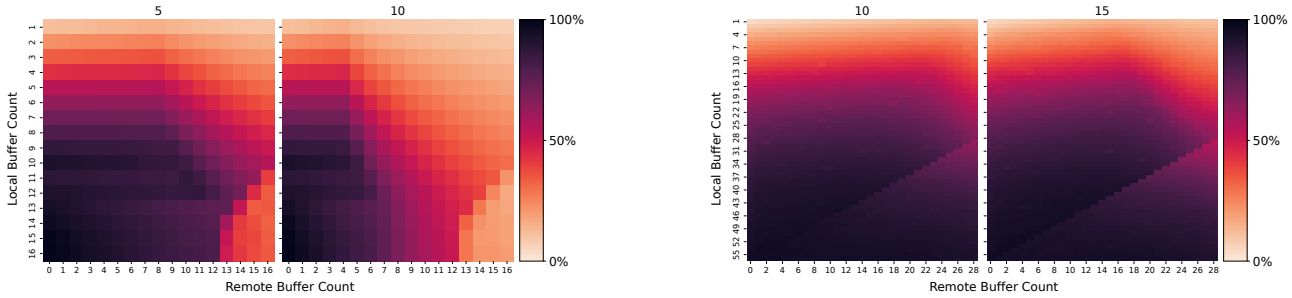
Operator Types. To gain better insights into the effects of the hardware from the perspective of in-memory data processing, we selected typical query operators from modern database engines [4, 14] – as the prime representative for in-memory data processing. We divide our operators into two categories: (i) operators with trivial internal state and (ii) operators with complex internal state. For the trivial internal state operators, we decided to employ the classes of *aggregation* and *filter* operations. Both operations exhibit similar properties, like a completely sequential data access on a single column and therefore, only the results of the aggregation kernel are reported. As an example for the aggregation, the *sum* is chosen as it mostly depends on the access speed for the data. The operators were equally executed in the following manner. A number of threads is spawned; for each thread, exactly one buffer filled with integer values between 0 and 100 is created. Every buffer exceeds the cache size to avoid caching-effects. All threads execute their respective operator on their own buffer. While the local buffer holds 100 % of the data, the remote buffers are sized according to the remote-to-local-bandwidth ratio. Generally, we report relative performance derived from the system bandwidth calculated as the sum of all data sizes of the used buffers divided by the time taken from starting all threads until the last one is finished, i. e., the synchronized wallclock time.

Contrary to the trivial state operators, we also implemented the *hash join* as a computationally heavier operator to represent the category of complex state operators. This operator works on two columns and involves random access to the hash table. Again, a number of threads is spawned, but this time for each thread there is one large table in local memory – the probe-table – and up to 10 smaller tables for creating the lookup-tables, on remote memory. Therefore, each thread has to perform a sequence of joins where the larger local table is joined against the smaller remote table.

3 Evaluation of Trivial State Operators

We start our evaluation by looking at using either 50 % (cf. Figure 2a) or 100 % (cf. Figure 2b) of the available physical cores for local work and up to 8 threads for remote work. These experiments highlight the scaling behavior with an increasingly larger amount of offloaded data for a given interconnect type. Henceforth UPI denotes the link between the NUMA nodes on the CXL hardware prototype and UPI2 refers to the link on the RDMA-supporting setup.

For each of the four experimental setups UPI, CXL, UPI2 and RDMA, the results are normalized to their respective value at full local work and therefore, at 0 remote buffers. Thus, 100 % for UPI and CXL is a different absolute value than 100 % for UPI2 and RDMA. This is mainly due to the fact that the experiments for UPI2/RDMA had to be executed on a different machine than the ones for UPI/CXL. In all setups, the remote buffer size that fits their respective bandwidth best is applied. Due to this setup, one line being above or beneath another does not mean the absolute performance of this technique is better or worse. In Figure 2, the results for the usage of 50 % and 100 % of the available physical cores for the work on local buffers are shown. For 50 % (cf. Figure 2a) the overall system bandwidth for UPI increases to a certain degree, when more workers are added to process remote buffers. RDMA on the other hand decreases while UPI2 holds



(a) RDMA - scaling remote data with 5 % and 10 % of local data.

(b) CXL - scaling remote data with 10 % and 15 % of local data.

Figure 3. Comparing the relative aggregation bandwidth trend for different interconnect types with varying amounts of local and remote data buffers and adjusted remote data size in percent.

the performance of full local work. For CXL, the curve looks similar to the one for UPI and thus, implies a similar scaling behavior, which is significantly better than the one shown by RDMA. For 100 % (cf. Figure 2b) UPI, CXL and UPI2 behave similarly, while being able to hold the performance of full local work. Contrarily, RDMA is dropping fast throughout the experiment because of hyperthreading effects. At this point, some of the processing threads and RDMA infrastructure threads are scheduled on the same physical core and thus induce triple usage of these cores and therefore, a performance decrease. This is highlighted even more by the mostly constant curve of UPI2 on the same machine. These figures show that while RDMA is capable of mostly holding its performance if there are resources available, it does not make sense to use it in high-workload-scenarios. UPI and CXL on the other hand scale well and offer the possibility to increase the system performance.

As these figures only show a very selected view of the whole search space, we offer a more complete overview for RDMA (cf. Figure 3a) and CXL (cf. Figure 3b) in the form of heatmaps. These show the impact of quantity and size of remote buffers by reporting the relative bandwidth, compared to the local maximum per heatmap. Figure 3 shows each combination for local (rows) and remote buffers (columns) for RDMA, i. e., up to 16 each, and from 1 to 56 local and 0 to 28 remote buffers for CXL, with varying remote data sizes.

The heatmaps of Figure 3a show the relative performance gains on the RDMA hardware for 5 % and 10 % remote buffer size compared to the fixed size local buffer. With the right fraction of remote data, it is possible to stay roughly at the same level of full local work. But when exceeding the optimal amount of remote data, the overall performance drops significantly as can be seen in the right heatmap of Figure 3a that has an overall lighter color. As the threads that work on the remote buffers are mostly waiting, their influence by hyperthreading is not that large. The very light spot in the lower right of each heatmap is a remarkable occurrence, which clearly indicates the triple usage of one core, i. e., the RDMA-communication-threads, local and remote working

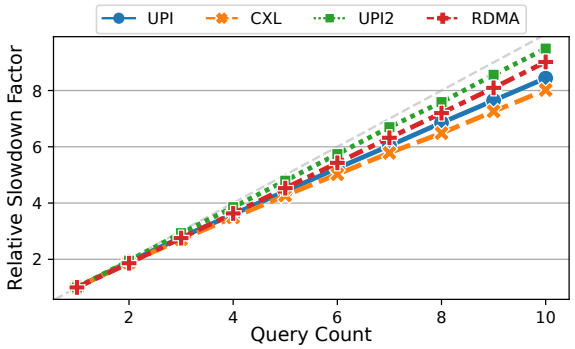


Figure 4. Relative behavior of UPI, CXL, UPI2 and RDMA with additional remote buffers for the hash-join kernel. Using 100 % of the available physical cores.

threads. This leads to the conclusion that RDMA can be used for offloading parts of the data, but its characteristics limit its scalability in this use-case.

Figure 3b in comparison shows the same experiment for CXL but different results. The results clearly show that the following two conclusions can be drawn: (i) As long as there are physical cores available for processing remote buffers (all rows except 56), we can either gain relative system bandwidth or at least do not lose out on any. (ii) The staircase pattern shows that a combination of remote and local buffer processing yields approximately the same performance as the next step. Therefore, we can conclude that it is possible to work on data located on CXL memory without sacrificing performance while gaining memory capacities. The lower right corner of each heatmap staircase pattern exhibits an overall lower bandwidth, which is because there are no physical cores left for processing and thus, hyperthreads are used, meaning double work for the physical core on which this hyperthread is scheduled. This effect becomes increasingly prevalent with larger remote buffers, as there is also more work to do for the hyperthread. This observation is independent of the actual data placement. Additionally, there is a shift to the left of the dark part of each heatmap visible with

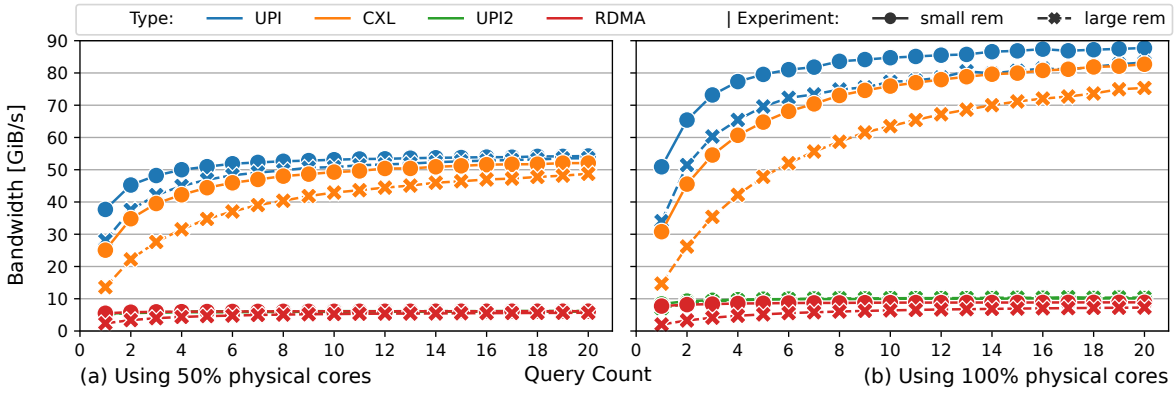


Figure 5. Absolute behavior of UPI, CXL and RDMA with additional remote buffers for the hash-join kernel applying pipeline grouping and prefetching strategies.

increasing remote data size, which is caused by the bandwidth of the CXL device. Thus, it is clear that the offloaded amount of data is a key performance factor. Moreover, it indicates that CXL scales with its available bandwidth and therefore, increases the portion of efficiently offloadable data.

4 Evaluation of Complex State Operators

As already introduced (cf. Section 2), we also evaluated the *hash join* as a complex state operator and show the results in Figure 4, again with adjusting the remote table size according to the relative bandwidth discrepancy. The relative system bandwidth for the other experiments does not make sense for the hash-join because the amount of work of each thread varies according to the number of joins. Thus, we report the relative slowdown compared to a single hash-join with increasing number of joins that are executed sequentially. The very naïve assumption would be a constant factor following the "twice the work, twice the time" principle (depicted as dashed, grey line), but this is not quite the case. In Figure 4, again the results for 100 % of the physical cores working in parallel on individual buffers is displayed. Even for the complex state operator, we observe that UPI and CXL behave similarly and RDMA cannot keep up with their scaling behavior. Counter-intuitively, UPI2 seems to perform worse than RDMA, which is due to the normalization. Each curve is again normalized to its performance for only one join. While all approaches with direct synchronous data access can start immediately, RDMA has to wait on the first data package and thus, the first join for RDMA is slow and the subsequent joins can profit from explicit prefetching. Remarkable is the fact that all approaches are below the assumed slowdown. This happens because with an increasing amount of executed joins, the actual compute portion and local random access into the hash table becomes the limiting factor and hardware acceleration mechanisms like prefetching are able to hide a lot of the introduced latency for retrieving the data from the remote source. Due to the asynchronous character of RDMA, it cannot benefit from these mechanisms that well. The plots for 50 % physical core usage show similar tendencies and are thus omitted.

5 Evaluation of Advanced Techniques

Our initial memory expansion experiments with naïvely using RDMA and CXL showed, that expectedly the network is indeed a performance bottleneck. While CXL is able to match the scaling performance of UPI, especially RDMA does not scale well in the shown benchmarks. To overcome that, we have developed a technique called pipeline grouping [5], which aims to efficiently overlap processing and network transfer. We want to build upon this principle and investigate the implications of explicit data access grouping for data that resides in a coherent address space. To facilitate that, we changed the access pattern for our UPI/UPI2 and CXL experiments from a synchronous to an asynchronous pattern. Meaning the data is explicitly copied from the remote source to local memory for further usage. This allows us to group the execution of multiple hash joins together and trigger data transfer much earlier, allowing for an efficient prefetching while reducing the network traffic by reducing redundant data transfers. However, this comes with the cost of pinning some threads - 5 in our case - for executing the copying operation from the remote source to local memory.

For a better overall impression, the following benchmark features up to 20 sequential hash joins, as introduced before. The used small columns are distinct to each other, while one large column is used for all of the 20 joins. Therefore, it can be seen as up to 20 individual pipelines on the same base column. This benchmark is executed in two variations: (i) We place the smaller columns remote and the large column is placed in local memory and (ii) the placements are inverted.

The results of this benchmark are reported in Figure 5. For the usage of 50 % of the available physical cores as depicted in Figure 5a, it is visible that again UPI and CXL behave similarly. The throughput difference between the UPI and

CXL curves is easily explained with the limited bandwidth of our prototypical CXL hardware setup. We are confident that this gap will shrink dramatically, when final CXL products are hitting the market. It is also visible that with a higher number of queries the gap between all 4 curves grows smaller. This is because more data can be reused and the absolute data transfer is reduced and therefore, the limited bandwidth has a lesser impact. RDMA and UPI2 on the other hand, perform significantly worse than UPI and CXL. At first sight, this seems counter-intuitive to our evaluation of pipeline groups [5], but going more into detail, instead of reducing the redundant data transfer in parallel executions, we applied it to sequential joins. Therefore, RDMA cannot benefit that well, as the amount of simultaneously requested data is not reduced and thus, its lower bandwidth and higher latency is a more dominating factor than for UPI and CXL. In addition to that, the local memory bandwidth of the UPI2/RDMA machine is significantly lower than the one of the UPI/CXL machine, which flattens the UPI2/RDMA curves even more. Furthermore, while all benchmarks need additional threads for the explicit data transfer (i. e., to force the asynchronous prefetching according to [5]), the utilization of the RDMA communication threads is much higher compared to UPI and CXL and thus, influences the performance of the executing physical cores much stronger. For the usage of 100 % of the available physical cores, shown in Figure 5b, the behavior is very similar, only the absolute values are larger. It is visible that the bandwidth limitation, especially for RDMA, has a higher influence for lesser query counts.

Overall, this benchmark has shown that, even though direct access to the CXL memory is possible, it can be beneficial to have an explicit asynchronous transfer from the CXL memory to local memory. If data access grouping is applied, it is less important which data is placed on what memory region, but rather if synchronous or asynchronous access shall be leveraged.

6 Related Work

Hardware disaggregation is by no means trivial and also a hot contemporary topic as presented in [20]. In a lot of cases, the naïve usage of disaggregation-strategies such as RDMA or CXL provide usually a higher latency and less throughput than state-of-the-art server architectures due to a longer physical distance. From our point of view, the most promising approaches for using RDMA are operator push-down as implemented in Farview [10] and pipeline grouping as introduced by us in [5] and applied in this paper. Both approaches offer appropriate techniques to reduce the network transfer significantly by either filtering data before transferring or avoiding redundant data transfers priorly.

However, as comprehensively evaluated in this paper and discussed in [7], CXL is the better solution from the perspective of in-memory data processing. Because of this, there is a lot of research ongoing with hardware disaggregation using CXL. On the one hand, there are several papers [2, 6, 12, 19, 21] discussing the possibilities CXL offers, from simple memory pooling to memory sharing and communication through shared memory and how to use it to bypass limitations of state-of-the-art-server architectures. Where [21] for example offers an approach to combine CXL-enabled memory and SSDs at virtually no performance cost to reduce the TCO further. On the other hand, papers like [7, 11, 16] laid their focus on the evaluation of early CXL devices in different scenarios, similar to this paper. With [16] providing the first absolute numbers of a CXL protoype device in microbenchmarks as well as real-world scenarios and [11] using CXL as expansion for the in-memory database system SAP HANA and applying typical database benchmarks as TPC-C and TPC-H, there are studies of CXL as memory expansion. While [3, 15] propose an approach for pushing down some computation to the CXL memory device to reduce the network-bottleneck; however, to the best our knowledge, none of them tried an explicit data-transfer model as we did in this paper. Therefore, it remains a future field of research to combine these approaches which should be very interesting, especially with CXL-enabled FPGAs.

7 Conclusion

Efficient in-memory data processing relies on the availability of sufficient resources, be it CPU time or available main memory. In order to meet the increasing resource demands, hardware disaggregation is a promising concept for the future. In line with ongoing research in this direction, we presented a comprehensive evaluation of the implications of different memory expansion methods (UPI, RDMA, and CXL) from the perspective of in-memory data processing in this paper. While recent work on RDMA has seemingly been a promising way out of over-provisioning of memory resources and high costs, in this paper we reproduced its limitations regarding scalability for memory expansion. In addition, we were able to show that – for representative operators from in-memory database engines and despite having an early CXL prototype hardware – (i) CXL yields comparable behavior to UPI and outperforms RDMA and (ii) CXL scales well. Hence, we see CXL as the most promising memory expansion technique for real world applications in the domain of in-memory data processing.

Acknowledgments

This work was partly funded by (1) the German Research Foundation (DFG) via a Reinhart Koselleck-Project (LE-1416/28-1) and (2) the European Union's Horizon 2020 research and innovation program under grant agreement No 957407 (DAPHNE). We would like to thank our colleagues Oleg Struk, Otto Bruggeman and Suprasad Mutalik Desai from Intel for their support with the CXL machine setup.

References

[1] Minseon Ahn, Andrew Chang, Donghun Lee, Jongmin Gim, Jungmin Kim, Jaemin Jung, Oliver Rebholz, Vincent Pham, Krishna T. Malladi, and Yang-Seok Ki. 2022. Enabling CXL Memory Expansion for In-Memory Database Management Systems. In *DaMoN@SIGDMO*. 8:1–8:5.

[2] Daniel S. Berger, Daniel Ernst, Huaicheng Li, Pantea Zardoshti, Monish Shah, Samir Rajadnya, Scott Lee, Lisa Hsu, Ishwar Agarwal, Mark D. Hill, and Ricardo Bianchini. 2023. Design Tradeoffs in CXL-Based Memory Pools for Public Cloud Platforms. *IEEE Micro* 43, 2 (2023), 30–38. https://doi.org/10.1109/MM.2023.3241586

[3] David Boles, Daniel Waddington, and David A. Roberts. 2023. CXL-Enabled Enhanced Memory Functions. *IEEE Micro* 43, 2 (2023), 58–65. https://doi.org/10.1109/MM.2022.3229627

[4] Peter A. Boncz, Martin L. Kersten, and Stefan Manegold. 2008. Breaking the memory wall in MonetDB. *Commun. ACM* 51, 12 (2008), 77–85.

[5] Andreas Geyer, Alexander Krause, Dirk Habich, and Wolfgang Lehner. 2023. Pipeline Group Optimization on Disaggregated Systems. In *CIDR*.

[6] Andreas Geyer, Daniel Ritter, Dong Hun Lee, Minseon Ahn, Johannes Pietrzyk, Alexander Krause, Dirk Habich, and Wolfgang Lehner. 2023. Working with Disaggregated Systems. What are the Challenges and Opportunities of RDMA and CXL?. In *BTW*, Vol. P-331. 751–755.

[7] Donghyun Gouk, Miryeong Kwon, Hanyeoreum Bae, Sangwon Lee, and Myoungsoo Jung. 2023. Memory Pooling With CXL. *IEEE Micro* 43, 2 (2023), 48–57. https://doi.org/10.1109/MM.2023.3237491

[8] Tim Kiefer, Benjamin Schlegel, and Wolfgang Lehner. 2013. BTW, Vol. P-214. 185–204.

[9] Thomas Kissinger, Tim Kiefer, Benjamin Schlegel, Dirk Habich, Daniel Molka, and Wolfgang Lehner. 2014. ERIS: A NUMA-Aware In-Memory Storage Engine for Analytical Workload. In *ADMS@VLDB*. 74–85.

[10] 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 *CIDR*.

[11] 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 *DaMoN@SIGMOD*. 35–43.

[12] 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, Volume 2* (Vancouver, BC, Canada) *(ASPLOS 2023)*. Association for Computing Machinery, New York, NY, USA, 574–587. https://doi.org/10.1145/3575693.3578835

[13] Iraklis Psaroudakis, Tobias Scheuer, Norman May, Abdelkader Sellami, and Anastasia Ailamaki. 2016. Adaptive NUMA-aware data placement and task scheduling for analytical workloads in main-memory column-stores. *PVLDB* 10, 2 (2016), 37–48.

[14] Mark Raasveldt and Hannes Mühleisen. 2019. DuckDB: an Embeddable Analytical Database. In *SIGMOD*. 1981–1984.

[15] Joonseop Sim, Soohong Ahn, Taeyoung Ahn, Seungyong Lee, Myunghyun Rhee, Jooyoung Kim, Kwangsik Shin, Donguk Moon, Euiseok Kim, and Kyoung Park. 2023. Computational CXL-Memory Solution for Accelerating Memory-Intensive Applications. *IEEE Computer Architecture Letters* 22, 1 (2023), 5–8. https://doi.org/10.1109/LCA.2022.3226482

[16] Yan Sun, Yifan Yuan, Zeduo Yu, Reese Kuper, Ipoom Jeong, Ren Wang, and Nam Sung Kim. 2023. Demystifying CXL Memory with Genuine CXL-Ready Systems and Devices. *arXiv preprint arXiv:2303.15375* (2023).

[17] The CXL Consortium. 2019. Compute Express Link. https://www.computeexpresslink.org/. [Online; accessed 14-December-2022].

[18] The CXL Consortium. 2019. Compute Express Link. https://www.computeexpresslink.org/projects-3. [Online; accessed 14-December-2022].

[19] Jacob Wahlgren, Maya Gokhale, and Ivy B. Peng. 2022. Evaluating Emerging CXL-enabled Memory Pooling for HPC Systems. In *2022 IEEE/ACM Workshop on Memory Centric High Performance Computing (MCHPC)*. 11–20. https://doi.org/10.1109/MCHPC56545.2022.00007

[20] 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. *PVLDB* 16, 1 (2022), 15–22.

[21] Qirui Yang, Runyu Jin, Bridget Davis, Devasena Inupakutika, and Ming Zhao. 2022. Performance Evaluation on CXL-enabled Hybrid Memory Pool. In *2022 IEEE International Conference on Networking, Architecture and Storage (NAS)*. 1–5. https://doi.org/10.1109/NAS55553.2022.9925356

[22] Hao Yu, José E. Moreira, Parijat Dube, I-Hsin Chung, and Li Zhang. 2007. Performance Studies of a WebSphere Application, Trade, in Scale-out and Scale-up Environments. In *IPDPS*. 1–8.