Programming the Future: the Essential Role of System Topology Awareness in Heterogeneous Disaggregated Environments

Beatrice Branchini
Politecnico di Milano, Italy
Pacific Northwest National Laboratory, USA
beatrice.branchini@polimi.it

Ian Di Dio Lavore
Politecnico di Milano, Italy
Pacific Northwest National Laboratory, USA
ian.didiolavore@pnnl.gov

Vito Giovanni Castellana
Pacific Northwest National Laboratory, USA
vitogiovanni.castellana@pnnl.gov

Marco Santambrogio
Politecnico di Milano, Italy
marco.santambrogio@polimi.it

ABSTRACT

Heterogeneous disaggregated systems represent a promising solution to deliver the performance required by next-generation High-Performance Computing (HPC) workloads. Nevertheless, their heterogeneity represents a significant challenge in the application development process, making it urgent to identify solutions to program these systems productively and efficiently without sacrificing performance. In this paper, we explore the potential of system topology information in addressing such a challenge. We delve into the foundations of system topology and how this information could be exploited in high-level programming libraries across all layers of the software infrastructures, from runtime systems to user-facing APIs. We propose K-Nearest Neighbors (KNN) as a simple case study, which we implement with a distributed programming library prototyped to exploit topology awareness. We demonstrate the solution's effectiveness on a commodity cluster with GPU-equipped nodes and illustrate how it will apply to next-generation disaggregated hardware.

CCS CONCEPTS

• **Hardware** → *Buses and high-speed links*; *Emerging technologies*; • **Computer systems organization** → *Distributed architectures*; • **Computing methodologies** → *Distributed programming languages*.

KEYWORDS

Disaggregated Memory, High-Performance Computing, Distributed Systems, Heterogeneous Computing, GPU, Programming Libraries

1 INTRODUCTION

The ever-increasing demand for computational power pushes modern High-Performance Computing (HPC) systems toward including increasingly complex hardware. Current generation HPC systems feature thousands of nodes, commonly including multicore Central Processing Units (CPUs) and accelerators, such as Graphics Processing Units (GPUs) or Field Programmable Gate Arrays (FPGA). Besides commodity coprocessors, which are here to stay, the landscape of next generation(s) hardware will be much more variegated, with systems equipped with diverse components such as Tensors Processing Units, memory expansions, or even custom chiplets to address the different needs of different workflows (Figure 1).

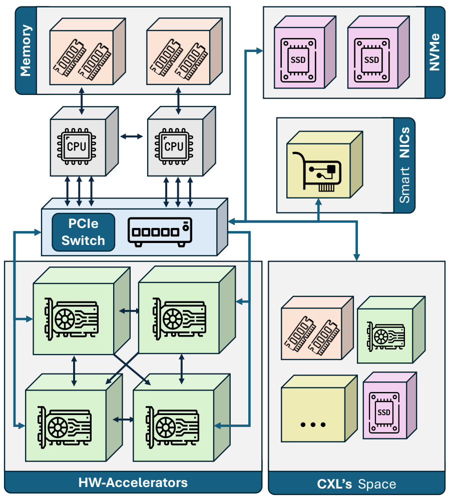

Figure 1: Modern systems are inherently heterogeneous.

In disaggregated systems, memory represents a significant concern for performance and programmability since different components have their physical memory and, typically, their own address space. In this context, Compute eXpress Link (CXL) represents a promising approach to connect CPUs, accelerators, and memory expansions, delivering high-performance memory coherency across different memory spaces [31]. Multiple specifications of the CXL protocol, built on top of the PCIe physical layer, have been proposed in recent years, with various vendors offering their implementations for their devices [32]. In the latest release (3.x), CXL enables multi-level switching designs with disaggregated memories over sophisticated network topologies [1].

Despite the promising approach, it is still not possible to practically deploy systems following such a specification [2, 3, 13], especially considering that most vendors currently support only early versions of CXL (i.e., 1.1) [32]. With time, devices compliant with more recent CXL specifications will become available; nevertheless, the lack of high-level environments for programming disaggregated systems could become a deterrent to early adoption. The infrastructure of such environments can be built on top of the Operating System (OS) architecture, runtime systems, programming library, or a combination of all of them [2]. In any case, in heterogeneous settings, the implementation of common operations such as *memory allocation* must incorporate features such as memory-tiering. Indeed, multiple levels of disaggregated memory regions will be available for allocation, with significant trade-offs with respect to size, latency, and overall bandwidth. Also, *addressing* becomes non-trivial. Based on how and where data is allocated, OS and library-level abstractions must incorporate complex memory mappings between virtual and physical addresses (OS abstraction) or sophisticated indexing schemes hidden underneath the implementation of smart pointers, iterators, and/or containers access methods (library/runtime abstraction). Furthermore, although shared-memory abstractions are convenient, *locality* must still be carefully considered. On disaggregated systems, such abstractions might inadvertently generate accesses to far-away memory regions, resulting in performance degradation due to high latency. Finally, memory disaggregation is not only due to component heterogeneity but could also result from the complex design of a single device, pushing the overall complexity even further. A typical example are CPUs, which may differ in Non-Uniform Memory Access (NUMA) configurations even within the same product family, significantly impacting applications' performance. Systems with different NUMA configurations expose different challenges, and extensive knowledge of the underlying hardware is required for efficient application development.

Some of the challenges associated with memory disaggregation are already attacked, with various techniques, on existing commodity clusters. The State of the Art proposes a multitude of libraries based on Partitioned Global Address Space (PGAS) models for distributed containers, and techniques such as active messages and work migration are well known to improve locality by moving computation where data reside. However, nodes in these systems are typically homogeneous (i.e., each node has the same hardware configuration), making it relatively easier for developers to reason about locality. Nevertheless, performance portability still represents an issue. Performance-critical applications are optimized for specific hardware, e.g., with custom thread affinity and process binding configurations based, for example, on NUMA properties. Porting these codes on multiple machines is intuitively not naive and often involves numerous trial-and-error cycles, contributing to the increased programming effort.

For all these reasons, we anticipate that *system topology* awareness will play a crucial role in the effective programming of performance portable applications, and we advocate for the need for topology information across all layers of the complex software stack, from runtime systems up to user-facing APIs.

Among the options discussed in [2], we further advocate that the first abstractions to offer adequate support for heterogeneous disaggregated systems will be libraries and runtime systems. We acknowledge that the changes required in existing OSs architectures are substantial and will require a tremendous engineering effort, especially for production-ready solutions. On the contrary, libraries and runtime systems are relatively easier to maintain and adapt to novel paradigms, shortening time-to-prototype and early adoption.

In this paper, we support our proposition by first highlighting the relevant topology properties necessary to operate on heterogeneous disaggregated architectures effectively; then, we demonstrate the practicality of our proposal by presenting a topology discovery mechanism integrated within the runtime layer of a high-level C++ library for distributed systems, SHAD [10]. We finally showcase how the topology information can be exploited in the upper layers of the stack to implement workloads on heterogeneous hardware.

2 SYSTEM TOPOLOGY PROPERTIES

In this Section, we highlight through motivating examples the information required by high-level libraries and runtime systems to successfully adapt to future heterogeneous disaggregated HPC systems. The research community has shown interest in defining standards for heterogeneous systems to integrate topology information within de-facto standard programming languages for the HPC domain. Exemplary of this are the several C++ proposals that discuss in a technology-agnostic way those properties [8, 9, 28]. Recognizing those efforts, we focus on pragmatically identifying a subset of those properties to be directly integrated into a topology discovery system.

Complex System Topologies. Once devices comply with the latest standards of CXL and become available for server-grade infrastructures, increased diversity of system topologies is expected, and the monolithic architecture of modern HPC clusters and cloud environments will likely not hold true anymore [3]. *Memory stranding*, i.e., the memory that cannot be rented out in cloud environments since no more CPU cores are available, can reach up to 25% of the overall capacity of a node [20]. Combining this aspect with the fact that memory counts to 40% of rack-level costs [22] and half of node-level costs [27], it appears evident that properly increasing the overall utilization is essential for cost-effectiveness. CXL empowers a paradigm shift that disaggregates the physical location of memory, I/O devices, and accelerators using the pool concept, i.e., the possibility to partition those resources on-demand, creating ad-hoc allocations for the end-users [16]. Studies have analyzed the capabilities of CXL to surpass the performance of classical RDMA approaches [22, 32], with the latest 3.0 and 3.1 specifications allowing peer-to-peer networking with direct access to the device's memory. It is also worth noting that modern clusters and supercomputers already feature a variety of interconnection structures spanning from tree-based topologies to mesh, ring, or spine/leaf architectures [3, 4, 6]. Therefore, incorporating the inner design of modern runtime systems and libraries topology information (e.g., **distance metrics** between memories/accelerators/compute nodes and **network topology** graphs) is paramount to accelerate workloads in future generations of disaggregated HPC architectures.

Heterogeneous Memories. Modern HPC systems comprise various kinds of memories, making them already heterogeneous with respect to overall system memory. Each device can be characterized through several metrics, among others **latency**, **bandwidth**, and **capacity**. Intuitively, these are required to achieve optimal data distributions. For example, a significant portion of datacenter applications' memory working set can be offloaded to *slower* memories [19, 22, 23]. CXL's tiered memory pooling [16] offers those applications a way to achieve this offloading procedure successfully. Moreover, several research works have highlighted the benefits of considering different types of non-volatile memories that are often underutilized, such as NVMe and Intel Optane™ drives [18, 33].

When it comes to memory disaggregation, virtually any large-scale application could benefit from topology information; however, big data analytics frameworks are among the ones that offer more opportunities for improvement. An example of such frameworks, GEMS, is described in [11, 12, 26]. GEMS is an in-memory graph processing engine optimized for executing SPARQL queries on large RDF datasets on commodity HPC clusters. The RDF format represents a graph as a collection of *subject-predicate-object* triples, with each predicate component associated with a Uniform Resource Identifier string. When an RDF dataset is ingested, GEMS creates a *Dictionary* which associates strings to numerical identifiers, and vice versa. These IDs are then used to populate the *Graph* data structures. Since RDF datasets could reach a trillion-tuple scale, these containers might reach massive memory footprints. However, it is important to notice that, most often, SPARQL queries rarely need frequent access to the Dictionary since constant labels appearing in a query could be looked up at once at the beginning of execution. Hence, with topology information, one could distribute dictionaries on slower, high-capacity memories. Similarly, caches for frequent terms could be allocated in faster memories, either closer or directly on compute nodes responsible for execution. Furthermore, in the presence of multiple graphs, topology awareness could allow storing them in different regions of the disaggregated memory based on, for example, their size or how frequently they are used.

Hardware Accelerators. If not the first, GPU acceleration is the most common example of system disaggregation, where pools of GPUs are made available to multiple servers to improve overall utilization and reduce costs [2]. Countless applications use hardware acceleration, such as PASTIS [30], a distributed pipeline for high-performance large-scale protein similarity search, i.e., identifying similarities across protein sequences. Given the computational intensity of the main task, that is, the actual alignment of the sequences, the authors integrated ADEPT [5] into the framework to considerably speed up the analysis [29]. ADEPT is a library that exploits GPUs to align genomic sequences, and it can run on devices from different vendors and, consequently, capabilities. On the single node, this library also offers the possibility to exploit multiple GPUs at the same time. For the author to extend PASTIS, topology information appears crucial to enable the use of GPUs, or accelerators in general. Firstly, the **availability** and the **number** of the accelerators within a node is essential to enable offloading the computation. Furthermore, **vendor-specific identifiers** are necessary to enable the use of the proper runtime and ensure portability across systems with different hardware. Finally, information about the device **computing capabilities** can help the end user to adapt the software to the underlying hardware quickly. In this context, examples of device-specific data for GPUs are the size of the off-chip and constant memory, the number of streaming multiprocessors, and the availability of Tensor Cores or native DPX instructions [14]. Similarly, for FPGAs, functional parameters are the maximum achievable clock frequency and memory information.

Summary. Different workloads may require different topology information to exploit disaggregated systems best. We identify the most relevant ones and classify them based on high-level classes of devices and spatial properties:

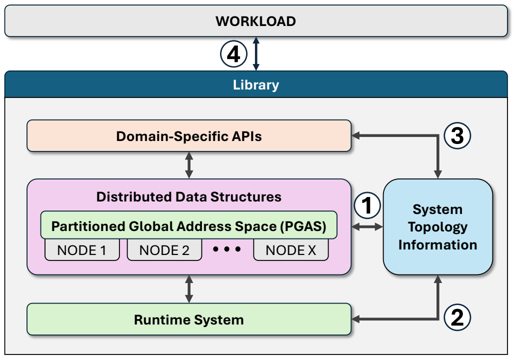
WORKLOAD
4
Library
Domain-Specific APIs
3
Distributed Data Structures
Partitioned Global Address Space (PGAS)
NODE 1
NODE 2
NODE X
1
System Topology Information
Runtime System
2

Figure 2: Components inside modern libraries targeting distributed systems can exploit topology information at different levels.

- Hardware accelerators:
  - Availability on a specific locale;
  - Vendor-specific identifier (to set the proper runtime);
  - Computing capabilities;
- Memories:
  - Availability on a specific locale;
  - Vendor-specific identifier;
  - Type;
  - Hardware features (e.g., size, latency, bandwidth, volatility/persistency);
- Processor:
  - NUMA domains characteristics;
  - Number of cores;
  - Support for Simultaneous Multithreading (SMT);
  - Thread enumeration;
- Spatial organization:
  - Distance between memories/accelerators/compute units/network interfaces;
  - How memories relate to hardware accelerators;
  - Network topology-graph;

3 ENABLING TOPOLOGY AWARENESS

As highlighted in the previous Sections, system topology information needs to be considered a central component in the design of libraries and frameworks for future disaggregated heterogeneous HPC systems. Topology information can be exploited at different stack levels when implementing the software infrastructure, as highlighted in Figure 2. This information must certainly be considered to build, maintain, and access high-performance distributed data structures (Figure 2, ①). Indeed, information like data locality is paramount to optimize data-processing kernels, possibly moving computation tasks where, or close to, the data resides. This will be even more prominent with CXL's `cxl.mem` devices, which might offer different performance levels based on the underlying architecture and their physical location within the disaggregated system. Additional information like NUMA-layout, hyperthreading capabilities, and process bindings should be exploited by the runtime system (Figure 2, ②) [21, 25, 34]. It has become common to identify domain-specific specialization of agnostic high-level libraries

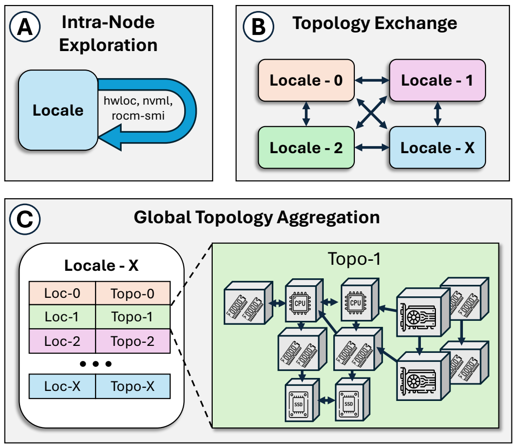

Figure 3: Schematic representation of the topology discovery pipeline we implement inside SHAD.

in fields such as Machine Learning or Graph Analytics [15, 24]. Properly exploiting architectural and topological knowledge (Figure 2, ③) of the complete distributed system while performing domain-specific operations has more potential than in agnostic solutions (e.g., exploiting Tensors Cores or DPX instructions based on their availability [17]). Finally, application developers might want to fine-tune their codes further based, for example, on domain knowledge of usage patterns. Therefore, proper APIs should expose the topology information (Figure 2, ④). In the following, we describe an exemplary implementation inside a high-level C++ library of how system topology information can be gathered and built up to provide a complete characterization of the system.

3.1 A Practical Example

To support our proposition, we implement a topology-aware framework on top of SHAD [10], an open-source C++ library for productive programming of distributed-memory systems. We consider it a representative of the libraries/runtimes that could be extended to support the broader class of disaggregated systems by incorporating topology awareness in their implementation and APIs. SHAD exposes a shared-memory programming interface, with APIs and semantics close to standard C++ (e.g., STL iterators, algorithms, access methods). Data is distributed across locales, i.e., computational units equipped with memory (e.g., a core, a NUMA domain, or a node), in such a way that each locale *owns* a portion of the data. Currently, data is distributed evenly across all the available locales. However, with topology information, this could be easily extended to change the distribution, such as allocating data on accelerator-equipped locales or memory expansions for rarely accessed data. Furthermore, SHAD can expose system topology information directly to the user-facing APIs to allow them to tune data distribution and execution policies (e.g., allocate data only on GPU-equipped nodes and process it on such accelerators). At a high level, we model the topology of the system as a map replicated on each locale for fast lookup. Each entry's key refers to a specific locale, and the value is its topology descriptor, which characterizes CPUs and hardware accelerators with availability and computing capabilities information. The topology is modeled as a graph to capture the interconnect scheme of the different system components conveniently.

Figure 3 depicts the steps of the topology discovery pipeline carried out automatically at run time when program execution starts. In Step Ⓐ, i.e., the *Intra-Node Exploration*, each system locale characterizes the hardware visible to them. In our current implementation, we leverage `hwloc` to build a hierarchical view of the system [7]. First, we record the CPU's NUMA domain configuration and core count, together with the processor's cache hierarchy. We then build an inventory of accelerators, if any. The accelerator descriptors report high-level information, such as their identifier, type (e.g., GPU, or FPGA), the off-chip memory size, and the interconnect with the CPU. Step Ⓑ performs the *Topology Information Exchange*. In this phase, each locale broadcasts the information about its own topology to all the other locales in the system. This information is then aggregated in Step Ⓒ, reconstructing the overall system topology. Finally, we propose an API to query the topology directory, e.g., to identify locales equipped with a specific device.

4 EXPERIMENTAL RESULTS

4.1 Experimental Setup

We validate our proof-of-concept implementation on a HPC cluster with 24 nodes. Each node has two Intel Xeon E5-2670, with 768 GB of main memory. The nodes are interconnected with a flat Infiniband network (Mellanox MT27500). While most commodity clusters have homogeneous nodes, in this system, instead, only five nodes of the available 24 are equipped with GPU accelerators (NVIDIA P100 with 16 GB High-Bandwidth Memory (HBM) memory). We leverage the cluster's heterogeneity to illustrate the benefits of exploiting topology awareness in high-level libraries.

4.2 Experimental Evaluation

We consider the exact K-Nearest Neighbors (KNN) problem as a use case. Given a *query vector* and a *dataset*, the KNN search aims at identifying the K vectors in the database that are *closer* to the query one. In our case, we compute the distance using the Euclidean distance as metric. We consider a dataset of 200k N-dimensional floating point vectors generated randomly, with N set to 2048. We execute a total of 10k queries for each run and set K to 100 nearest neighbors for each query. Starting from the properties discussed in Section 2, we leverage a few exemplary ones to highlight the necessity of topology awareness. In particular, we exploit the availability of hardware accelerators and their memory profiles. Based on the availability of accelerators in the cluster, we measure the application performance with three different configurations, namely `CPU-only`, `CPU+GPU`, and `topo-aware GPU-only` (Figure 4).

Figure 5 reports the strong scaling evaluation of the execution time in seconds for all three configurations. We collect the execution times of 10 iterations and average the results using arithmetic mean. The first configuration, i.e., `CPU-only`, is a naive implementation where the computation is scheduled only on the CPU, regardless of the availability of accelerators. The `CPU-only` implementation leverages the PGAS distributed containers available in the SHAD library. Execution time in the `CPU-only` is reported as the red line in Figure 5. We report performance improvements when increasing the number of nodes as expected for a parallel workload.

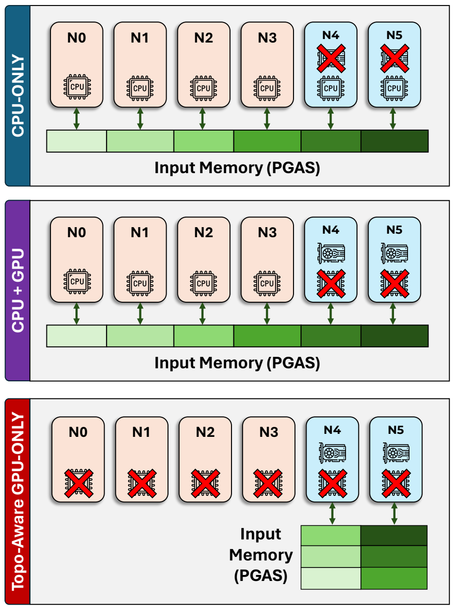

Figure 4: Test configurations for our workload, with increasing levels of exploited system topology (top to bottom)

The second configuration is the CPU+GPU version. This configuration does not consider topology information when distributing data, as in the previous case. However, it does consider GPU availability, i.e., each locale involved in the computation offloads processing to the fastest available compute unit (CPU/GPU). As a result, accelerator-equipped locales offload computation on the GPU, while all the other only use the CPU cores. The green line in Figure 5 depicts the scaling properties of this configuration. The first five nodes all feature a GPU, and since we schedule the computation on the fastest compute unit, execution time improves and scales with the number of nodes. However, further including CPU-only nodes brings diminishing returns due to the high unbalance. In this scenario, since data is evenly distributed, and each locale processes the same amount of data, overall latency is dominated by the CPU-only nodes' contribution, voiding the performance benefits of acceleration. Hence, for a number of nodes greater than 5, performance follows the same trend as the CPU-only with the slowest locales (i.e., the non-accelerated ones), bounding the overall achievable performance.

Finally, the last configuration, i.e., the topo-aware GPU-only, exploits topology information in both stages when distributing and scheduling the computation. Given a heterogeneous distributed system, we filter the nodes and select only those connected to a hardware accelerator. This operation is done at runtime without prior knowledge of the system organization. In this case, once the nodes with a GPU are identified, we allocate data only on those nodes, increasing the PGAS locality in GPU nodes. We report the scaling of the execution time for this configuration as the blue line in Figure 5. Increasing the number of nodes improves performance as long as accelerators are added. Then, data from non-accelerated locales is partitioned across the accelerated ones. Thus, locales performing the computations are the ones equipped with a GPU.

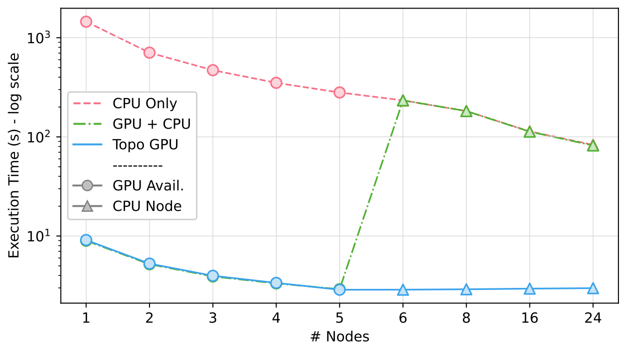

Figure 5: Scaling study of the KNN execution time from 1 to 24 nodes, exploiting different levels of topology information.

This translates into a plateau since adding CPU-only nodes does not contribute to the processing.

These three configurations highlight how efficiently exploiting topology information can improve performance. Hence, including such information and exposing it to the end user can improve memory management and performance. The proposed approach is general enough to be easily extended to much more sophisticated hardware with significantly more complex application scenarios.

5 CONCLUSIONS

Heterogeneous disaggregated systems promise to deliver the performance demanded by next-generation workloads. Understanding and leveraging their structure is paramount for users to exploit them efficiently. This paper advocates for including system topology information in modern programming libraries to achieve superior performance portability on heterogeneous disaggregated hardware. We present a practical example by integrating system topology awareness into the runtime and data structures layers of a C++ library for distributed systems. We also analyze how leveraging this information at different levels affects application performance. While our current proof-of-concept implementation only supports a limited number of devices, in future work, we will investigate more sophisticated disaggregated designs, including memory expansion nodes with potentially limited compute capabilities.

ACKNOWLEDGMENTS

This research is based upon work supported by the Office of the Director of National Intelligence (ODNI), Intelligence Advanced Research Projects Activity (IARPA), through the Advanced Graphic Intelligence Logical Computing Environment (AGILE) research program, contract number DE-AC05-76RL01830. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the ODNI, IARPA, or the U.S. Government.

REFERENCES

[1] Ishwar Agarwal. 2022. CXL overview and evolution. In *Proc. Hot Chips*. 1–24.

[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 Operating Systems Review* 57, 1 (2023), 38–46.

[3] Hasan Al Maruf and Mosharaf Chowdhury. 2023. Memory disaggregation: advances and open challenges. *ACM SIGOPS Operating Systems Review* 57, 1 (2023), 29–37.

[4] Mohammad Alizadeh and Tom Edsall. 2013. On the Data Path Performance of Leaf-Spine Datacenter Fabrics. In *2013 IEEE 21st Annual Symposium on High-Performance Interconnects*. 71–74.

[5] Muaaz G Awan, Jack Deslippe, Aydin Buluc, Oguz Selvitopi, Steven Hofmeyr, Leonid Oliker, and Katherine Yelick. 2020. Adept: a domain independent sequence alignment strategy for gpu architectures. *BMC bioinformatics* 21 (2020), 1–29.

[6] Maciej Besta, Jens Domke, Marcel Schneider, Marek Konieczny, Salvatore Di Girolamo, Timo Schneider, Ankit Singla, and Torsten Hoefler. 2020. High-performance routing with multipathing and path diversity in ethernet and hpc networks. *IEEE Transactions on Parallel and Distributed Systems* 32, 4 (2020), 943–959.

[7] François Broquedis, Jérôme Clet-Ortega, Stéphanie Moreaud, Nathalie Furmento, Brice Goglin, Guillaume Mercier, Samuel Thibault, and Raymond Namyst. 2010. hwloc: A generic framework for managing hardware affinities in HPC applications. In *2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*. IEEE, 180–186.

[8] Gordon Brown, Ruyman Reyes, Michael Wong, H. Carter Edwards, Thomas Rodgers, Mark Hoemmen, Patrice Roy, Carl Cook, Jeff Hammond, Hartmut Kaiser, Christian Trott, Paul Blinzer, Alex Voicu, Nat Goodspeed, and Tony Tye. 2018. P0796r3: Supporting Heterogeneous & Distributed Computing Through Affinity. https://api.semanticscholar.org/CorpusID:211222533

[9] Gordon Brown, Ruyman Reyes, Michael Wong, H. Carter Edwards, Thomas Rodgers, Mark Hoemmen, and Tom Scogland. 2020. P1436r2: Executor properties for affinity-based execution. https://api.semanticscholar.org/CorpusID:211483209

[10] Vito Giovanni Castellana and Marco Minutoli. 2018. SHAD: The Scalable High-Performance Algorithms and Data-Structures Library. In *2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. 442–451.

[11] Vito Giovanni Castellana, Marco Minutoli, Shreyansh Bhatt, Khushbu Agarwal, Arthur Bleeker, John Feo, Daniel Chavarría-Miranda, and David Haglin. 2017. High-performance data analytics beyond the relational and graph data models with gems. In *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 1029–1038.

[12] Vito Giovanni Castellana, Alessandro Morari, Jesse Weaver, Antonino Tumeo, David Haglin, Oreste Villa, and John Feo. 2015. In-memory graph databases for web-scale data. *Computer* 48, 3 (2015), 24–35.

[13] Nan Ding, Samuel Williams, Hai Ah Nam, Taylor Groves, Muaaz Gul Awan, LeAnn Lindsey, Christopher Daley, Oguz Selvitopi, Leonid Oliker, and Nicholas Wright. 2022. Methodology for Evaluating the Potential of Disaggregated Memory Systems. In *2022 IEEE/ACM International Workshop on Resource Disaggregation in High-Performance Computing (REDIS)*. IEEE, 1–11.

[14] Anne C Elster and Tor A Haugdahl. 2022. Nvidia hopper gpu and grace cpu highlights. *Computing in Science & Engineering* 24, 2 (2022), 95–100.

[15] Matthias Fey and Jan Eric Lenssen. 2019. Fast graph representation learning with PyTorch Geometric. *arXiv preprint arXiv:1903.02428* (2019).

[16] Donghyun Gouk, Miryeong Kwon, Hanyeoreum Bae, Sangwon Lee, and Myoungsoo Jung. 2023. Memory pooling with cxl. *IEEE Micro* 43, 2 (2023), 48–57.

[17] Yufeng Gu, Arun Subramaniyan, Tim Dunn, Alireza Khadem, Kuan-Yu Chen, Somnath Paul, Md Vasimuddin, Sanchit Misra, David Blaauw, Satish Narayanasamy, and Reetuparna Das. 2023. GenDP: A Framework of Dynamic Programming Acceleration for Genome Sequencing Analysis. In *Proceedings of the 50th Annual International Symposium on Computer Architecture* (Orlando, FL, USA) *(ISCA '23)*. Association for Computing Machinery, New York, NY, USA, Article 25, 15 pages.

[18] Suhas Jayaram Subramanya, Fnu Devvrit, Harsha Vardhan Simhadri, Ravishankar Krishnawamy, and Rohan Kadekodi. 2019. Diskann: Fast accurate billion-point nearest neighbor search on a single node. *Advances in Neural Information Processing Systems* 32 (2019).

[19] Andres Lagar-Cavilla, Junwhan Ahn, Suleiman Souhlal, Neha Agarwal, Radoslaw Burny, Shakeel Butt, Jichuan Chang, Ashwin Chaugule, Nan Deng, Junaid Shahid, et al. 2019. Software-defined far memory in warehouse-scale computers. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. 317–330.

[20] 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.

[21] Marcos Maroñas, Antoni Navarro, Eduard Ayguadé, and Vicenç Beltran. 2023. Mitigating the NUMA effect on task-based runtime systems. *The Journal of Supercomputing* 79, 13 (2023), 14287–14312.

[22] 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, Volume 3* (<conf-loc>, <city>Vancouver</city>, <state>BC</state>, <country>Canada</country>, </conf-loc>) *(ASPLOS 2023)*. Association for Computing Machinery, New York, NY, USA, 742–755.

[23] Hasan Al Maruf, Yuhong Zhong, Hongyi Wang, Mosharaf Chowdhury, Asaf Cidon, and Carl Waldspurger. 2023. Memtrade: Marketplace for Disaggregated Memory Clouds. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 7, 2 (2023), 1–27.

[24] Tim Mattson, Timothy A Davis, Manoj Kumar, Aydin Buluc, Scott McMillan, José Moreira, and Carl Yang. 2019. LAGraph: A community effort to collect graph algorithms built on top of the GraphBLAS. In *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 276–284.

[25] Ugljesa Milic, Oreste Villa, Evgeny Bolotin, Akhil Arunkumar, Eiman Ebrahimi, Aamer Jaleel, Alex Ramirez, and David Nellans. 2017. Beyond the socket: NUMA-aware GPUs. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture* (Cambridge, Massachusetts) *(MICRO-50 '17)*. Association for Computing Machinery, New York, NY, USA, 123–135.

[26] Alessandro Morari, Vito Giovanni Castellana, Oreste Villa, Antonino Tumeo, Jesse Weaver, David Haglin, Sutanay Choudhury, and John Feo. 2014. Scaling semantic graph databases in size and performance. *IEEE Micro* 34, 4 (2014), 16–26.

[27] Timothy Prickett Morgan. [n. d.]. CXL And Gen-Z Iron Out A Coherent Interconnect Strategy — nextplatform.com. https://www.nextplatform.com/2020/04/03/cxl-and-gen-z-iron-out-a-coherent-interconnect-strategy/. [Accessed 11-06-2024].

[28] Thomas Rodgers, Patrice Roy, Carl Cook, Jeff R. Hammond, Hartmut Kaiser, Christian R. Trott, Paul Blinzer, and Alexandru Virgil Voicu. 2020. P1795r2: System topology discovery for heterogeneous & distributed computing. https://api.semanticscholar.org/CorpusID:211482877

[29] Oguz Selvitopi, Saliya Ekanayake, Giulia Guidi, Muaaz G Awan, Georgios A Pavlopoulos, Ariful Azad, Nikos Kyrpides, Leonid Oliker, Katherine Yelick, and Aydin Buluç. 2022. Extreme-scale many-against-many protein similarity search. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–12.

[30] Oguz Selvitopi, Saliya Ekanayake, Giulia Guidi, Georgios A Pavlopoulos, Ariful Azad, and Aydın Buluç. 2020. Distributed many-to-many protein sequence alignment using sparse matrices. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–14.

[31] Debendra Das Sharma. 2022. Compute Express Link®: An open industry-standard interconnect enabling heterogeneous data-centric computing. In *2022 IEEE Symposium on High-Performance Interconnects (HOTI)*. IEEE, 5–12.

[32] 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* (Toronto, ON, Canada) *(MICRO '23)*. Association for Computing Machinery, New York, NY, USA, 105–121.

[33] Shao-Peng Yang, Minjae Kim, Sanghyun Nam, Juhyung Park, Jin yong Choi, Eyee Hyun Nam, Eunji Lee, Sungjin Lee, and Bryan S. Kim. 2023. Overcoming the Memory Wall with {CXL-Enabled}{SSDs}. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*. USENIX Association, Boston, MA, 601–617.

[34] Jie Zhang, Xiaoyi Lu, and Dhabaleswar K. (DK) Panda. 2017. Designing Locality and NUMA Aware MPI Runtime for Nested Virtualization based HPC Cloud with SR-IOV Enabled InfiniBand. In *Proceedings of the 13th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments* (Xi'an, China) *(VEE '17)*. Association for Computing Machinery, New York, NY, USA, 187–200.