



OPEN
Compute Project



WHITE PAPER:

NEAR DATA COMPUTING, PRESENT & FUTURE

Version 1.0

Authors:

Yang Seok Ki, Samsung

Seungwon Lee, Samsung

Oscar Pinto, Samsung

Bill Martin, Samsung

Michael Choi, Samsung

Seong Kim, AMD

Chris Stillo, AMD

Jason Molgaard, Solidigm

Mats Oberg, Marvell

Jongryool Kim, SK hynix

Woosuk Chung, SK hynix

Om Rameshwar Gatla, SK hynix

Jon Hermes, ARM

Matt Bromage, ARM

Date: August 19, 2025

1 Version

Date	Version #	Author	Description
August 19, 2025	1.0	Jason Molgaard (workstream lead uploading) and others listed on title page	First published version from the FTI DCC workstream group

Current OCP Template Version:

Whitepaper/Document Submission Template v1.6.0

Effective August 2025

Date: August 19, 2025



This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

2 Executive Summary

While CPU performance improvement based on Dennard scaling and Moore's Law has already reached its limit, handling large amounts of data in a power-efficient manner requires re-examining the model for moving data from traditional storage to the CPU. This model consumes a lot of power and limits performance due to bandwidth limitations.

This whitepaper proposes Near Data Computing to address these challenges based on domain-specific computing and its architectures as an alternative to overcome the limitations of the existing CPU-centric computing model. Domain-specific computing, seen in early graphics and network cards, has expanded into accelerators such as GPUs, TPUs, FPGAs, and IDPs. Meanwhile, hyperscalers, where power efficiency is particularly important, use ASICs or FPGAs to offload and accelerate OS, security, and data processing tasks. As data is explosively generated in the cloud, machine learning, big data, and edge, the industry is actively reviewing the application of these technologies in various fields.

Date: August 19, 2025



This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

3 Compliance with Open Compute Project Tenets

3.1 Openness

This whitepaper promotes openness by advocating for standardized APIs, programming models, and interfaces for Near Data Computing technologies, enabling collaborative development and interoperability across vendors and communities in the OCP ecosystem.

3.2 Efficiency

The paper enhances efficiency by proposing Near Data Computing solutions that minimize data movement, reduce host CPU utilization, and optimize power consumption, leading to higher performance in data-intensive workloads.

3.3 Impact

The whitepaper drives impact by outlining innovative NDC technologies and use cases that challenge traditional CPU-centric models, influencing industry shifts toward domain-specific computing and accelerating adoption in AI, big data, and edge applications.

3.4 Scale

It addresses scale by focusing on hyperscale environments, such as cloud data centers handling massive data growth, and provides architectures like Computational Storage and DPUs that support parallel processing and scalability across large infrastructures.

3.5 Sustainability

This contribution supports sustainability by emphasizing energy-efficient computing paradigms that lower data center power usage—projected to rise dramatically—through reduced data transmission and offloading, aligning with goals like the Climate Neutral Data Center Pact for emission reductions by 2030.

4 Table of Contents

1	Version.....	2
2	Executive Summary	3
3	Compliance with Open Compute Project Tenets	4
3.1	Openness.....	4
3.2	Efficiency.....	4
3.3	Impact.....	4
3.4	Scale	4
3.5	Sustainability	4
4	Table of Contents.....	5
5	Why Near Data Computing	7
5.1	Introduction	7
5.2	Data Center Energy Consumption	7
5.3	Practical Application	10
6	Near Data Computing Technologies	12
6.1	Computational Storage.....	12
6.2	Processing-In-Memory	13
6.3	Processing-Near-Memory.....	14
6.4	DPU (Data Processing Unit)	15
6.5	Advantages of these technologies.....	16
6.6	Near Data Computing Applications Exemplar.....	16
7	Near Data Computing Use Cases	19
7.1	Computational Storage Use Cases	19
7.1.1	Key Value Computational Storage Drive.....	19
7.1.2	Search In Storage Use Case	22
7.1.3	Database Use Case	23
7.1.4	Video Analytics Use Case.....	25
7.1.5	Video Processing Use Case	25
7.1.6	Data Integrity Check	26
7.2	Processing-In-Memory Use Cases.....	28
7.2.1	One Memory-Coupled Computing (OneMCC).....	28

Date: August 19, 2025



This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

7.3	Processing-Near-Memory Use Cases.....	30
7.3.1	AI Framework Use Case.....	30
7.3.2	VectorDB Offload Use Case	33
7.4	Data Processing Unit (DPU) Use Cases	34
7.4.1	DPU Use Case.....	34
8	Layer Abstractions	35
8.1	Introduction	35
8.2	End User Interface.....	35
8.2.1	Near Data Computing Interface.....	35
8.3	Platform Developer Perspective	39
8.3.1	Programming Options.....	39
8.3.2	Platform Use Cases	43
9	Management	46
9.1	Redfish.....	46
9.1.1	Discovering with Redfish	47
9.1.2	Challenges with Redfish	48
10	Conclusion	49
11	Acronyms	50
12	License.....	51
13	About Open Compute Project.....	52

Date: August 19, 2025



This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

5 Why Near Data Computing

5.1 Introduction

Since the advent of the cloud, a large amount of data, in various forms, is being generated, at high speed. Big data technology, capable of efficiently storing and processing such data, is being developed and disseminated. Most of these technologies are open source, and have created healthy and rich communities. As a result, data storage and processing technologies have been democratized and large amounts of data have been easily produced and consumed.

The enormous amount of data we've never seen before, has created a new phenomenon. In 2010, Dave McCrory introduced the concept of Data Gravity: the application is inevitably pulled, by the gravity created by massive data. This means that moving computation is easier than moving data. As IO latency and throughput requirements increase, gravity gets stronger, because the time for data transfer gets shorter. In order to speed up data transfer times, an application or service gets closer and closer to the data.

We can observe this phenomenon in both data centers and edge. Before the cloud era, data warehouses were popular for large-scale data analytics, such as decision support and business intelligence applications. Since the advent of the cloud, however, a large amount of unstructured data and semi-structured data, as well as structured data has been generated. Recently, Data lake technology has helped store such data for many different analytic products and workloads, and that can effectively perform various data processing methods to interpret this data, making it scalable and flexible.

A good solution should handle massive data efficiently, in a power-efficient manner and apply data gravity all the way down to the device-level with domain specific architectures.

5.2 Data Center Energy Consumption

Data center electricity use is increasing dramatically, especially for AI. One study reveals that data center electricity use in Ireland has more than quadrupled since 2015. The percentage of total metered electricity consumption used by data centers increased from 5% in 2015 to 21% in 2023¹. Urban households accounted for 18% of total metered electricity consumption while data centers consumed a total of 30 TWh in 2023. It is forecast that these data centers may use a third of Ireland's energy supply by 2026. Denmark data center energy use is projected to rise from 1.1 TWh in 2021 to 7.8 TWh in 2030 and account for almost 15% of the country's total energy consumption². This is forecast to a tenfold increase of a total of 10 TWh by 2035. The increase in energy consumption of the service sector is mainly due to the construction of data centers which are characterized by having a consumption that is almost constant during all hours of the year.

The estimated global data center electricity consumption in 2022 was 240-340 TWh³, or 1-1.3% of total global electricity demand (excludes cryptocurrency mining). Figure 1 shows the data center energy consumption increase and the cost of data transmission over networks. Data centers have become more efficient over the years in the cloud and hyperscale environments with IT hardware and cooling. Despite these efficiencies, there has been substantial increase in energy use over the past several years by 20-40% annually due to rapid growth in workloads. It is expected that this trend

¹ Central Statistics Office, Ireland <https://www.cso.ie/en/statistics/energy/datacentresmeteredelectricityconsumption>

² The Danish Energy Agency https://ens.dk/sites/ens.dk/files/Basisfremskrivning/kf23_hovedrapport.pdf

³ International Energy Agency <https://www.iea.org/energy-system/buildings/data-centres-and-data-transmission-networks>

will continue over the next few years. The International Energy Agency (IEA) forecasts electricity usage from AI and data centers to double within the next four years.

Global trends in digital and energy indicators, 2015-2022			
	2015	2022	Change
Internet users	3 billion	5.3 billion	+78%
Internet traffic	0.6 ZB	4.4 ZB	+600%
Data centre workloads	180 million	800 million	+340%
Data centre energy use (excluding crypto)	200 TWh	240-340 TWh	+20-70%
Crypto mining energy use	4 TWh	100-150 TWh	+2300-3500%
Data transmission network energy use	220 TWh	260-360 TWh	+18-64%

Figure 1: Global trends in digital and energy indicators, 2015-2022

AI refers to different technologies and methods that enable intelligent machines. Generative AI which is used for creating new content, such as text-generating ChatGPT, text to images with OpenAI's DALL-E to name a few. Large datasets in terabytes of data may be fed at the training phase of these models which are considered most energy intensive. Hugging Face's BigScience Large Open-Science Open-Access Multilingual (BLOOM) module consumed 433 MWh of energy in its training phase⁴. While emphasis is given to the training phase on energy consumption, other scholars are pointing out that the inference phase may be equally concerning with Google reporting 60% of AI related energy consumption was from inference between 2019 to 2021⁵. The energy usage may not be similar for different LLMs and their phases, but is considerable and may need further analysis. The AI boom has led to an increased demand in AI chips leading to a significant rise in AI's energy footprint. Alphabet's chairman was quoted in February 2023 that LLM usage could "likely cost 10 times more than a standard keyword search".

⁴ Luccioni et al 2022, Estimating the Carbon Footprint of BLOOM <https://arxiv.org/abs/2211.02001>

⁵ De Vries 2023, The growing energy footprint of artificial intelligence, Joule

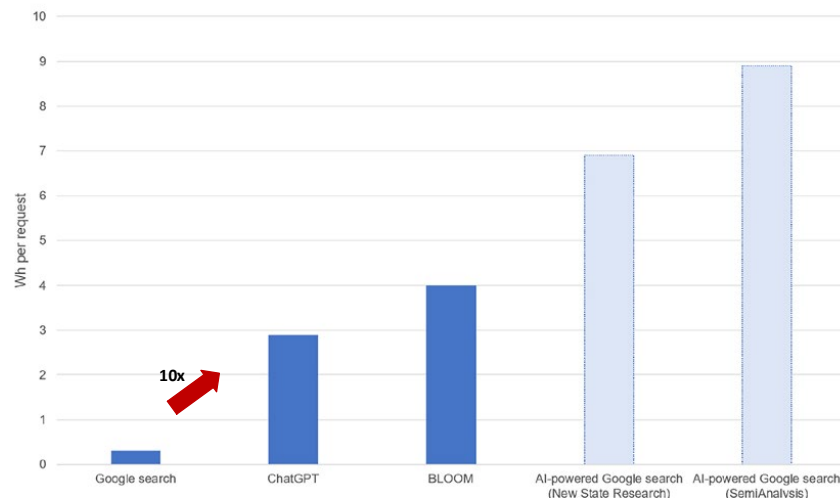


Figure 2: Estimated energy consumption for AI powered systems

The ICT sector has published a science based pathway for ICT emissions in line with the Paris Agreement, compatible with 1.5°C scenario. Data Centre Trajectory requires a 53% emission reduction by 2030⁶ to achieve the goal. In response to this request, the Climate Neutral Data Center Pact was launched and developed by industry on Jan. 21 2021⁷. Data centers and cloud infrastructure providers are working towards Climate Neutrality by 2030, with 5 metrics to achieve in 2025 and 2030. These 5 metrics include 1) Energy efficiency, 2) Clean energy, 3) Water conservation, 4) Circular economy and 5) Circular energy systems. Of these five areas, the suppliers of server components can make a direct contribution to two areas: the circular economy and the energy efficiency. Especially, this paper shares a vision, how a new computing paradigm can improve energy efficiency.

According to a paper published in 2020 by Science⁸, the number of servers used in data centers has increased dramatically since 2010 and servers and storage have become a major component of data center power consumption, accounting for more than 50% of electricity usage (Figure 3). This means that reducing server and storage power consumption is most critical to meeting the emission target of data centers.

⁶ Science Based Targets initiative (SBTi), Guidance for the ICT sector <https://sciencebasedtargets.org/sectors/ict>

⁷ Climate Neutral Data Center, Climate Neutral Data Center Pact <https://www.climateneutraldatacentre.net/downloads/>

⁸ Recalibrating global data center energy-use estimates
https://datacenters.lbl.gov/sites/default/files/Masanet_et_al_Science_2020.full_.pdf

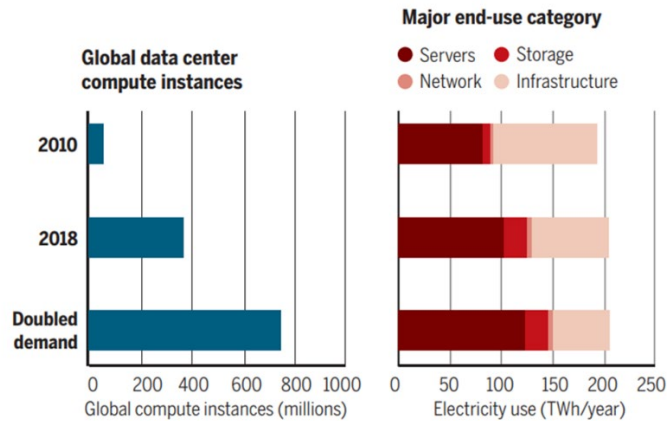


Figure 3: Data center electricity consumption

A server's overall computing power can be determined by the server's computing density and its utilization. If we can deliver more computing power with the same electric power budget, fewer servers can meet our computing needs. During the period when Dennard scaling and Moore's law are applied, the computational power can be doubled, every 18 months, while keeping the electric power consumption constant. The number of transistors per square millimeter increases as chip processing technology advances. As chip processing technology advances, it consumes less power for the same number of transistors. However, since the mid-2000s, Dennard scaling is no longer effective, due to the power wall, caused by leakage current and threshold voltage. And Moore's law also takes longer to increase transistor density.

Before the cloud became popular, servers were provisioned for peak times and underutilized in normal situations. In this case, the static power consumption of servers cannot be neglected when the server is not heavily used. Virtualization and cloud technologies have reduced this idle time and significantly improved overall server utilization. Over 80% of data centers rely on cloud technologies or use hyper-scalers. Over 92% of the servers are already virtualized. Given these facts, it is not realistic that server utilization can be significantly improved.

It is unfortunate that the existing technologies appear to have little room for further improvement in near future. Alternatively, we need to pay attention to domain-specific architectures, as Professor John Hennessy emphasized. According to Hwang's law, the number of transistors in GPUs is increasing at a faster rate than Moore's law. We notice that one of the major architectural issues that drove this paradigm shift was Carbon zero requirements for electric power. Various types of domain-specific architectures are booming, especially in the AI and network domains. However, domain-specific architectures still depend on and use data flows, as defined for general purpose compute.

5.3 Practical Application

What would be a good solution to handle massive data efficiently in a power-efficient manner? The data gravity can be applied all the way down to device-level with domain specific architecture. Applications have different performance characteristics. Let's take a closer look at domain specific computing architecture from a data state perspective. According to its state, data can be classified into data at rest, data in transit, and data in use. In the past, CPUs were used universally, regardless of data states. However, now there are many domain specific architectures, that can efficiently process the data in each state. The domain-specific architecture is purpose-built for power-efficient and high-performance data processing. First, data in use usually refers to data residing in volatile

memory and various NPUs, including GPUs and TPUs, utilize memory hierarchy, to process compute-intensive data efficiently. On the other hand, Data in Transit refers to data moving through a network. A SmartNIC, DPU, Smart Switch, etc., can effectively process network acceleration functions such as packet processing, protocol processing, and storage network functions, using Open Programmable Infrastructure (OPI)⁹. Finally, data at rest refers to persistent data residing on storage devices. Computational storage can effectively handle data transformation and filtering functions, while the data is stored to or retrieved from storage.

All of these technologies move functions to the device, which match the intrinsic capabilities of the device and are more optimized than how a CPU processes them. These technologies increase computing capabilities beyond the Von Neumann architecture that the traditional scale-up and scale-out solutions rely on. As a result, servers are more efficient and powerful, and eventually it is possible to reduce the number of servers required for a given workload.

Applying compute near data may be the right direction to follow to address the growing energy consumption in the data center. Depending of the workload and the applied compute solution, it may be possible to meet the energy guidelines for the future. This does require understanding the processing requirements and capabilities to meet these guidelines while improving overall system efficiency. The solutions must address scalability, lower power usage, and lower bandwidth footprint while meeting the aforesaid goals. This may involve defining hardware abstractions, programming models, and handling data parallelism by default. Thus, given the variety of processing units and capabilities that Near Data Computing (NDC) can provide, multiple models may be required. The gist is that by defining the hardware abstractions and programming models for NDC, vendors can decide which abstractions and programming models they support, and users can choose which programming models they support for their applications. This white paper addresses the lack of guidance that exists on which NDC technology should be used for specific types of applications and defines the current state of NDC development.

⁹ OPI is a community driven standards based open ecosystem for next generation architectures and frameworks based on DPU/IPU like technologies. Open Programmable Infrastructure project <https://opiproject.org/>

6 Near Data Computing Technologies

Near Data Computing (NDC) is the concept of providing compute capabilities near the data that the computation is to be performed on. There are three basic categories of NDC that are described in this white paper: Computational Storage, Processing-in-Memory, Processing-near-Memory, and DPUs. There are many terms that are being used in the industry that have the same meaning as these terms. The general term that refers to all three categories is Near Data Computing, which within the industry has also been referred to as Data Centric Computing. Computational Storage has been referred to in the industry as In Storage Compute and Storage Intelligence. Processing-In-Memory has been referred to in the industry as In-Memory-Processing and Computational Memory. Processing-Near Memory has been referred to as Near-Memory-Processing and also as Computational Memory.

NDC introduces an additional attack surface that requires security measures. These issues need to be considered when developing NDC environments.

6.1 Computational Storage

Computational Storage is a technique where there is compute on or near the storage device as shown in Figure 4. This architecture is described in the SNIA Computational Storage Architecture and Programming Model¹⁰.

Computational Storage is used when Compute is required on stored data. This means that for that data, Computational Storage:

- a) eliminates moving data that is to be computed on from the device to host memory, improving performance, power utilization, and resource availability;
- b) reduces host CPU utilization; and
- c) if the results of the computation are to be used by the host for other processing, only those results, which may be significantly less than the data being computed on, are moved to the host.

The SNIA Computational Storage Architecture and Programming Model specifies three types of Computational Storage Devices (CSx). As shown in Figure 4, these CSXes are a Computational Storage Processor (CSP), a Computational Storage Drive (CSD), and a Computational Storage Array (CSA). A CSP is a device that connects to the storage fabric and contains compute resources, but does not have its own storage. The CSP communicates with the associated storage device utilizing the storage fabric without host intervention. A CSD is a device that contains both storage and compute (e.g., an SSD with computational storage capability, or an HDD with computational storage capability). A CSA is a storage array that contains computational storage resources. The CSA may also include CSDs which would provide multiple levels of offload.

¹⁰ SNIA Computational Storage Architecture and Programming Model, <https://www.snia.org/csarch>

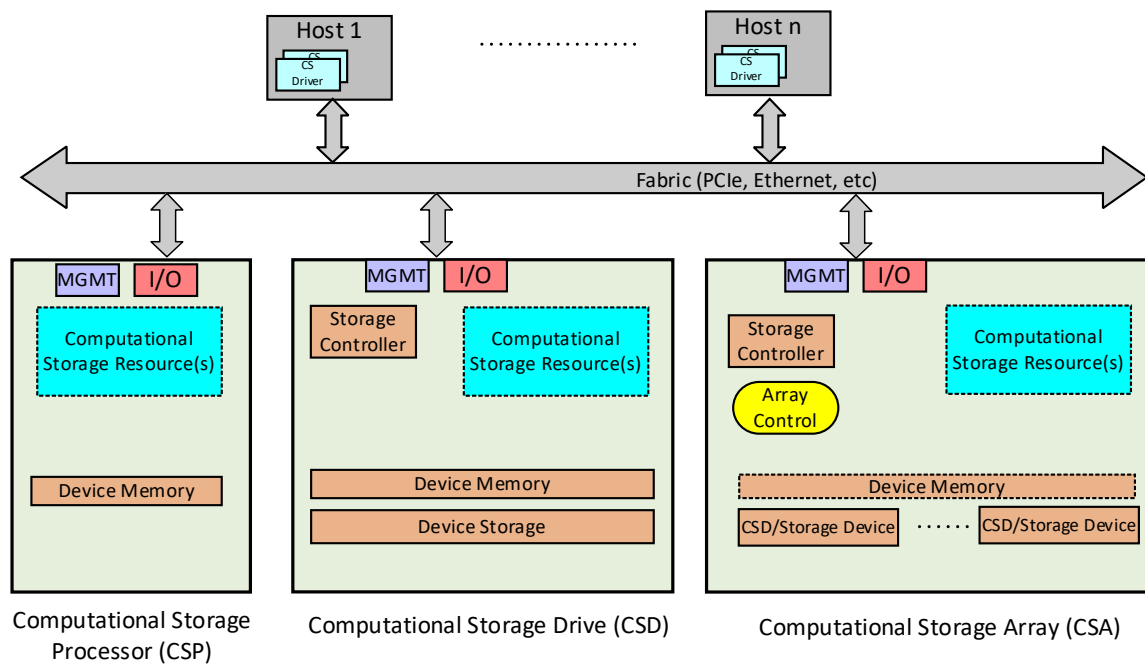


Figure 4 Computation Storage Devices (CSxes)

Computational Storage is used when data that is intended for compute is in a storage device or is destined to be stored in a storage device. Performing compute on the storage device offloads compute from the host CPU. For data that is already located on the storage device, this compute reduces data movement across a bus from the storage device to host memory. Additionally, if the results of the compute is destined for host memory, the amount of data transmitted over a bus to host memory is less than the amount transferred if the data to be computed on is moved into host memory for compute by the host CPU. If the output of the compute is to be stored on the storage device, then there is no data transfer following the compute. All of these benefits result in reduced power consumption, reduced latency, and improved performance.

6.2 Processing-In-Memory

Processing-In-Memory (PIM) is a technique where there is compute on the memory chip or on the memory card that is inserted in the system, as shown in Figure 5. The PIM technology leverages bank-level parallelism to expose high on-chip bandwidth of standard DRAM to processors. It can be located in any DRAM within a system and it is coupled with a bank. PIM can be seamlessly integrated with any commodity DRAM, such as DDR, LPDDR, HBM, GDDR DRAM, and more. The PIM device is tied to a memory bank to load data from Input Output Sense Amplifier (IOSA) and store data into the bank directly. The PIM device consists of Processing Element (PE) and register files. The PE consists of the control logic to execute instructions and the simple Multiply-Accumulate (MAC) unit to perform a calculation. The register file is used to load and store data for the MAC operations, and it also contains an instruction register as well. The current data precision supported by the PE of the PIM device is half-precision floating-point format.

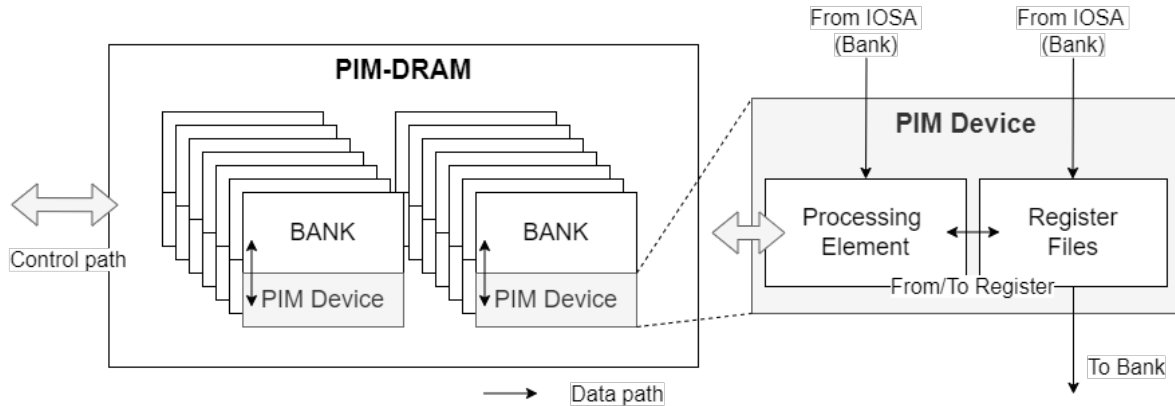


Figure 5 Processing-In-Memory

Processing-In-Memory is used for data that is in use and therefore is resident in memory. This means that for that data, Processing-In-Memory eliminates moving data that is to be processed across the memory bus, offloads host CPU utilization, and the results of the processing remain in the memory allowing use of the results without additional use of the memory bus. All of these benefits result in reduced power consumption, reduced latency, and improved performance.

6.3 Processing-Near-Memory

Processing-Near-Memory (PNM) is processing located near the memory allowing processing of data without requiring the host processor to move data into host memory. Unlike Processing-In-Memory, in PNM the processor is separate from the memory, but is also separate from the host CPU (see Figure 6). The advantage of PNM is that it offloads processing from the host CPU.

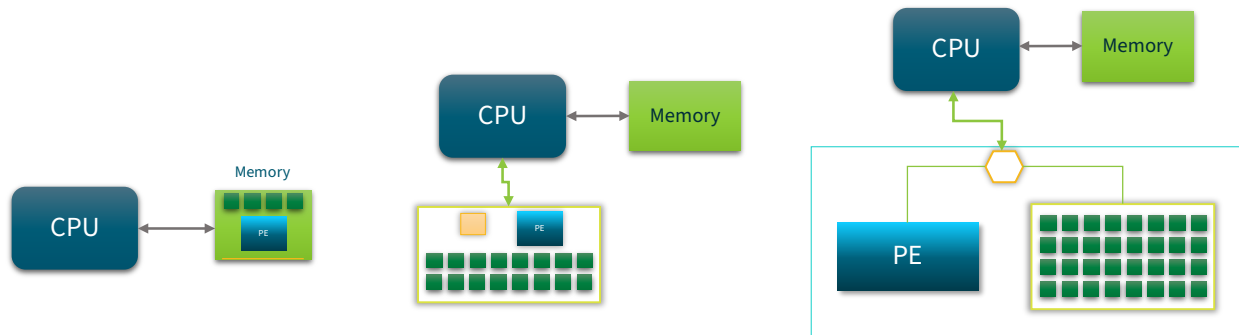


Figure 6 Processing-Near-Memory

The benefit in system designs containing PNM grows as tasks and applications become increasingly more memory-limited, be that from constraints based on memory access latency or growing bandwidth requirements to feed data into the machine.

We surveyed a large number of existing ML inferencing tasks with an eye toward their memory sensitivity. By inducing slowdowns in access time and limiting their memory bandwidth, we could observe which of those tasks were the most memory sensitive and thus the most likely to benefit from PNM acceleration. For this survey, we optionally slowed down all memory, only intermediate result memory, or only the weights and static data of a model. Figure 7 shows the variability of our

search: many models were ultimately insensitive to the location of weights, and the degree of sensitivity to the placement of their intermediate results was wide.

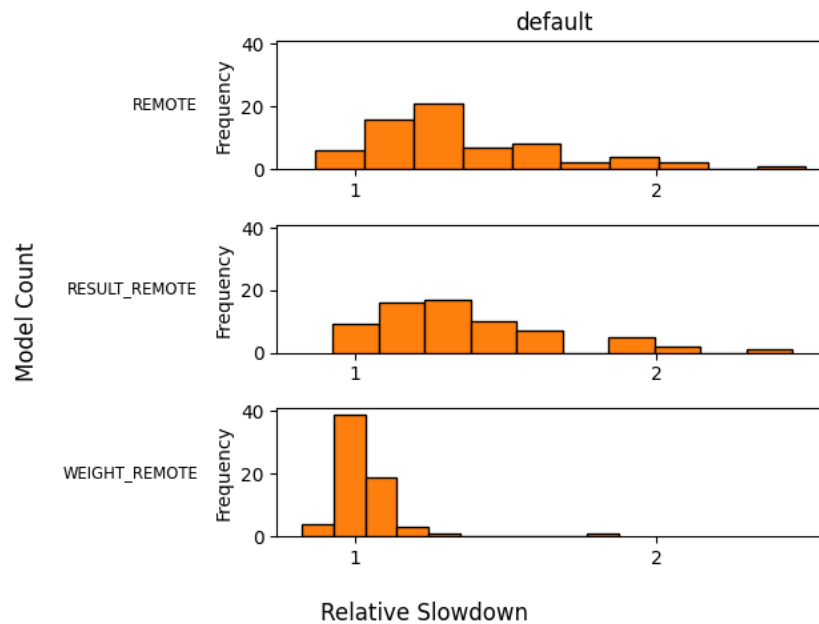


Figure 7 Measuring total runtime slowdown across many models when memory is selectively slowed

6.4 DPU (Data Processing Unit)

A DPU is an accelerator card designed to offload specific networking and security functions from the host CPU (see Figure 8). It typically supports a Direct Memory Access (DMA) paradigm, enabling efficient packet transmission and reception across the network. Additionally, DPUs may offer customizable acceleration functions, which can include networking protocols like RDMA over Converged Ethernet (ROCE), TCP offload, or domain-specific optimizations. These capabilities enhance overall system performance by freeing up CPU resources and providing granular networking and security services close to workloads. DPUs play a crucial role in modern data centers, improving workload density and enabling robust network services while maintaining application performance and security. All of these benefits result in reduced power consumption, reduced latency, and improved performance.

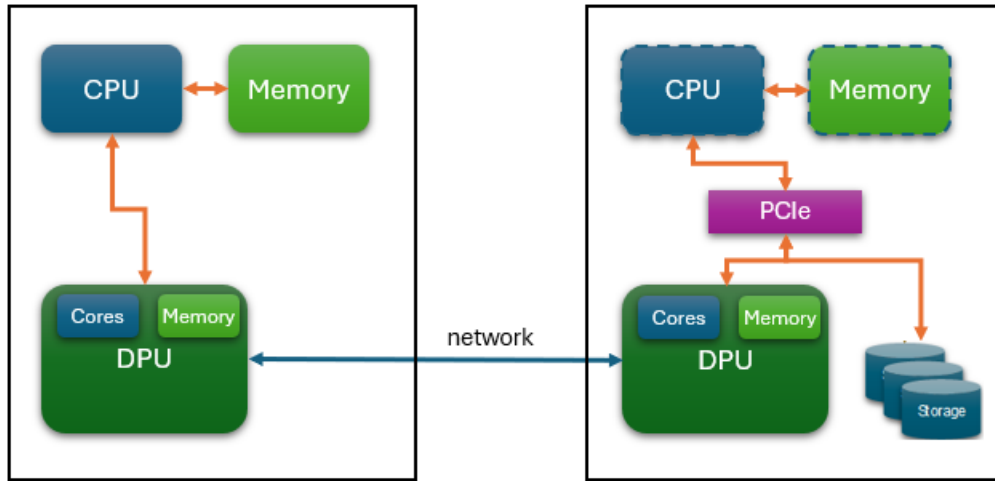


Figure 8 DPU

6.5 Advantages of these technologies

With the energy consumption reaching new records every year fueled by AI adoption and data generation, NDC technologies may be the solution to overcome increasing energy requirements. There is data available that shows the benefits of adopting these technologies at different points in the system as shown in Table 1. There are performance, energy efficiencies and latency benefits that may be derived from adopting them.

Technology	Performance	Energy Efficiency
In-Memory Compute ¹¹	10-23.2x	1.64-5.23x
Near Memory Compute ¹²	1.39-1.77x	31.6%
Data Processing Unit (DPU) ¹³	54x	24-29%
Computational Storage ¹⁴	4.3x	36%

Table 1: Near Data Computing Advantages

6.6 Near Data Computing Applications Exemplar

NDC may use any of the previously defined technologies. Table 2 show a list of example computation and potential locations for performing that computation. This list is not intended to be an exhaustive

¹¹ Juan Gomez-Luna et al, "Benchmarking Memory-Centric Computing Systems: Analysis of Real Processing-in-Memory Hardware" <https://arxiv.org/pdf/2110.01709>

¹² Liu Ke et al, "Near-Memory Processing in Action: Accelerating Personalized Recommendation with AxDIMM" <https://ieeexplore.ieee.org/abstract/document/9489313>

¹³ BlueField-2 DPU, "NVIDIA BlueField DPUs Drive Data Center Efficiency" <https://blogs.nvidia.com/blog/bluefield-dpus-energy-efficiency>

¹⁴ Mahdi Torabzadehkashi et al, "Computational storage: an efficient and scalable platform for big data and HPC applications" <https://journalofbigdata.springeropen.com/articles/10.1186/s40537-01>

list but a list of examples for the ability of the reader to understand why compute may be located in differing locations.

Application	Compute ¹	Discussion	Example ²
Encryption	In Memory, Near Memory	Encryption is done prior to moving data to storage Decryption is done after receiving data from storage	
	DPU	Used to encrypt data that is being stored on multiple devices where the management of the where the data is to be stored is done by a device on the network	
	Computational Storage	Used to decrypt data before performing some other computation and to encrypt data that is stored following a compute operation	
Compression	In Memory, Near Memory	Compression is done prior to moving data to storage decompression is done after receiving data from storage	
	DPU	Used to compress data that is being stored on multiple devices where the management of the where the data is to be stored is done by a device on the network	
	Computational Storage	Used to decompress data before performing some other computation and to compress data that is stored following a compute operation	3.1.3
Metadata creation	In Memory, Near Memory	For data that is located in memory, hash, indexes, etc. may be generated and stored in the memory location where the data is located.	
	Computational Storage	For data that is located in device storage, hash, indexes, etc. may be generated and stored back on the storage device.	3.1.1, 3.1.3
Data Search	In Memory, Near Memory	For data that is located in memory, that data may be searched and the generated results stored in the memory location where the data is located.	
	Computational Storage	For data that is located in device storage, that data may be searched and the generated results may be stored back on the storage device or may be transferred to host memory.	3.1.2, 3.1.1
Database manipulation	In Memory, Near Memory	For a database that is located in memory, hashes may be generated, filtering may be done, and data conversion may be performed and the generated results stored in the database.	
	Computational Storage	For a database that is located in device storage, hashes may be generated, filtering may be done, and data conversion may be performed and the generated results stored in the database.	3.1.3

Data Analytics	In Memory, Near Memory	For data that is located in memory, that data may be analyzed to determine some action to take based on that analysis and the results modify the data stored in memory.	
	DPU	For data that is in transit, that data may be analyzed to determine some action to take based on that analysis and the results modify the data that is in transit.	
	Computational Storage	For data that is located in device storage, that data may be analyzed to determine some action to take based on that analysis and the results modify the data that is stored.	3.1.4, 3.1.5
Data Integrity Check	Computational Storage	The integrity of data that is located in device storage helps to prevent data corruption.	3.1.6
1 Compute is the computational technology that is to be used on the data (i.e., In Memory Compute, Near Memory Compute, DPU, or Computational Storage) 2 Example is the section where an example use case is described. A blank in this column does not mean that there are no use cases, only that this document does not contain such a use case description.			

Table 2: Near Data Computing Application Location

7 Near Data Computing Use Cases

7.1 Computational Storage Use Cases

To create ultra-efficient storage solutions that meet enterprise and cloud server-class performance, dependability, and security requirements, a variety of platforms, including powerful x86 CPUs and optional accelerators, are required. These solutions may be built upon general purpose CPUs (e.g., x86) and FPGAs. These processors can unlock considerable gains in storage efficiency and FPGAs can provide higher performance with a better TCO for traditional storage architectures and next-generation software-defined storage networking environments.

Computational Storage Platform Options include: CSD; CSP; CSA; and external computational storage array remote storage. These different options can be used depending on the deployment environment and the storage configuration. The external computational storage array represents NVMe-oF™.

There are various acceleration functions for computational storage, including:

- a) basic data management task accelerations such as compression, encryption, parsing, and conversion;
- b) security accelerations, such as search-in-storage (see Figure 13) and homomorphic encryption;
- c) database accelerations for PostgreSQL (see Figure 14) and Greenplum (see Figure 15); and
- d) machine learning accelerations for video analytics and data analytics (see Figure 16 and Figure 17).

The use cases listed above can be called near data-compute. Specifically, compute capability is located right next to where the data is located. Therefore, the acceleration unit can read the data directly from the storage and process that data depending on the application. The accelerator can send the processed data to the host CPU or write the data back to the storage without host CPU intervention.

7.1.1 Key Value Computational Storage Drive

During data analytics, a scientist may be interested in only a small portion of a given dataset, leading to query patterns that can be highly selective and do not require reading back all data ^{15 16}. To speed up applications with potentially highly selective data access patterns, embedded key-value stores such as LevelDB ¹⁷ and RocksDB ¹⁸ have been increasingly explored in scenarios ranging from filesystem metadata management ¹⁹, block data management ²⁰, to largescale data analytics

¹⁵ J. Gu, S. Klasky, N. Podhorszki, J. Qiang, and K. Wu, "Querying large scientific data sets with adaptable io system adios," in Supercomputing Frontiers, R. Yokota and W. Wu, Eds., Cham: Springer International Publishing, 2018, pp. 51–69

¹⁶ S. Byna, J. Chou, O. Rübel, Prabhat, H. Karimabadi, W. S. Daughton, V. Roytershteyn, E. W. Bethel, M. Howison, K.-J. Hsu, K.-W. Lin, A. Shoshani, A. Useton, and K. Wu, "Parallel I/O, analysis, and visualization of a trillion particle simulation," in Proceedings of the 2012 International Conference on High Performance Computing, Networking, Storage, and Analysis (SC 12), 2012, 59:1–59:12. DOI: 10.1109/SC.2012.92.

¹⁷ Leveldb, <https://github.com/google/leveldb>, 2014.

¹⁸ Rocksdb, <https://rocksdb.org/>, 2014.

¹⁹ K. Ren, Q. Zheng, S. Patil, and G. Gibson, "IndexFS: Scaling file system metadata performance with stateless caching and bulk insertion," in Proceedings of the 2014 International Conference for High Performance Computing, Networking, Storage, and Analysis (SC 14), 2014, pp. 237–248. DOI: 10.1109/SC.2014.25.

²⁰ A. Aghayev, S. Weil, M. Kuchnik, M. Nelson, G. R. Ganger, and G. Amvrosiadis, "The case for custom storage backends in distributed storage systems," ACM Trans. Storage, vol. 16, no. 2, 2020. DOI: 10.1145/3386362.

²¹. Unfortunately, even though these stores have been increasingly deployed they are still limited in their capacity to process data efficiently.

SK hynix's Key Value Computational Storage Drive (KV-CSD) features:

- a) a Key-Value Interface that eliminates duplicate layers and fine-grained management (see Figure 9);
- b) an LSM Data Structure that offloads sorting data and building indexes into KV-CSD; and
- c) ZNS²² Optimization that improves Write Amplification (WAF) and removes Garbage Collection.

Additionally, the KV-CSD supports Range Query and Histogram for Analytics Query Acceleration. Figure 10 demonstrates the benefits of offloading secondary index. In traditional data analytics, a “Full Scan” operation is triggered to query data that satisfies a range of input attributes. This operation incurs significant read IO operation and consumes expensive host resources. Secondary index addresses this problem by maintaining a small secondary indexes over a range of data. KV-CSD offers an API to maintain secondary indexes on-device, resulting in saving compute and IO resources.

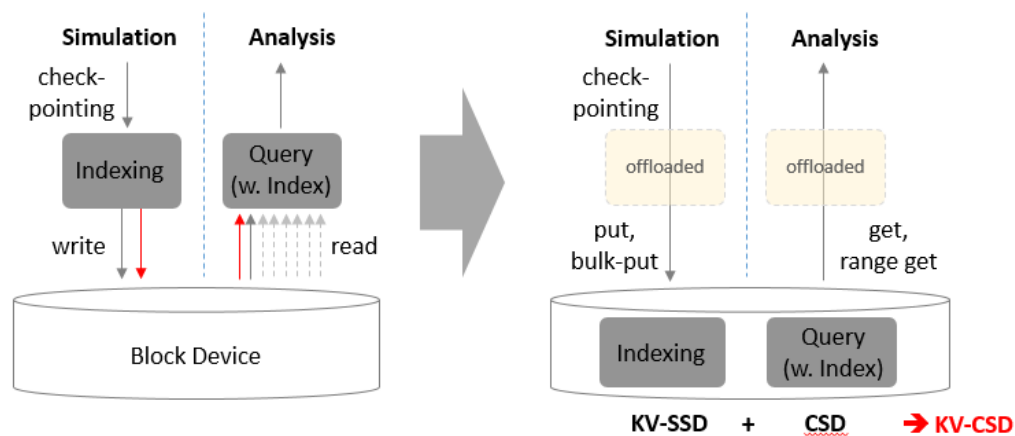


Figure 9: High Performance KV-CSD for Analytics

²¹ S. Alsubaiee, A. Behm, V. Borkar, Z. Heilbron, Y.-S. Kim, M. J. Carey, M. Dreseler, and C. Li, “Storage management in asterixdb,” Proc. VLDB Endow., vol. 7, no. 10, 841–852, 2014. DOI: 10.14778/ 2732951.2732958.

²² NVMe Zoned Namespaces (ZNS) Command Set Specification, <https://nvmexpress.org/specifications/#content-13264>

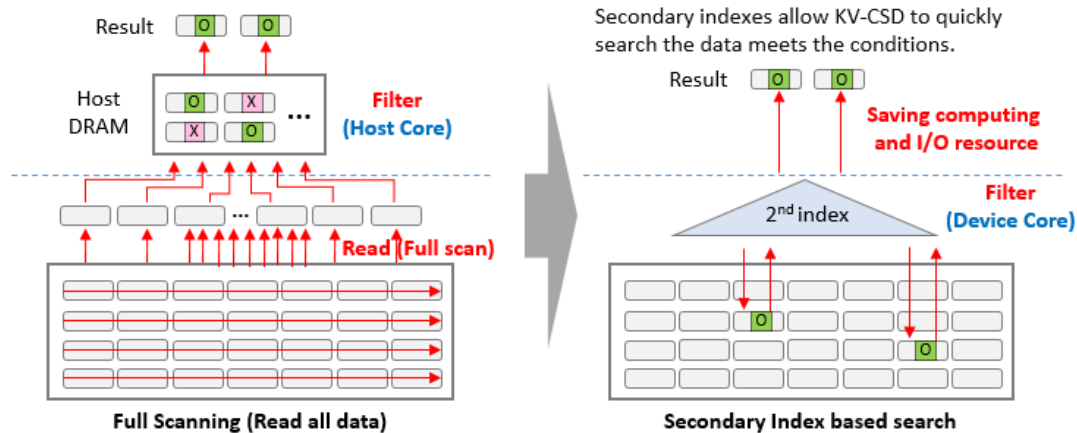


Figure 10 : Offloading Secondary Index with KV-CSD

Figure 11 and Figure 12 shows that KV-CSD improves query performance and achieves higher write performance with fewer host CPU cores respectively. By performing secondary index, KV-CSD is 7.4x faster than RocksDB (software-based KV Stores). Furthermore, KV-CSD reduces the total amount data written by the host, as it avoids the process of data compaction.

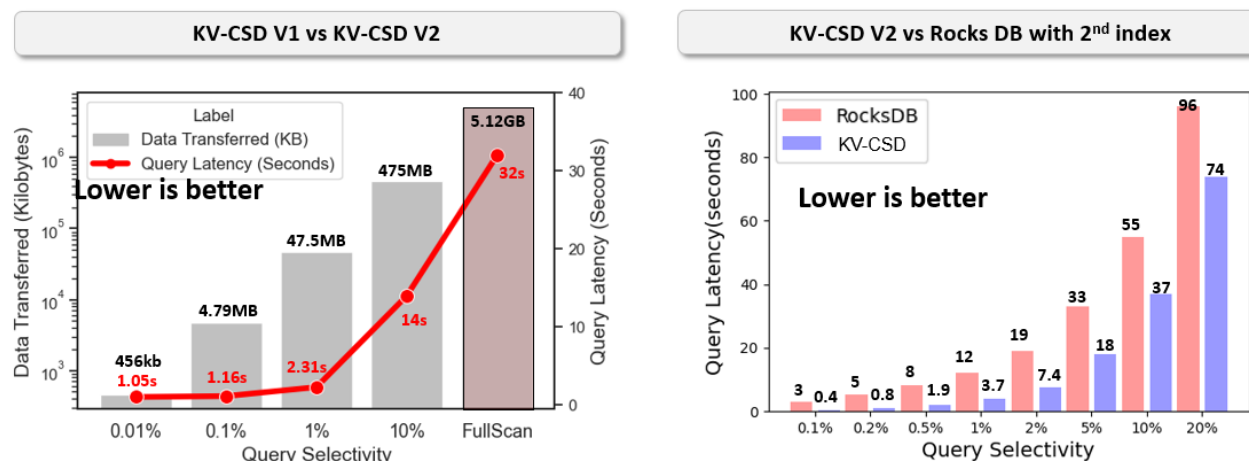


Figure 11 : Boosted Query Performance with KV-CSD

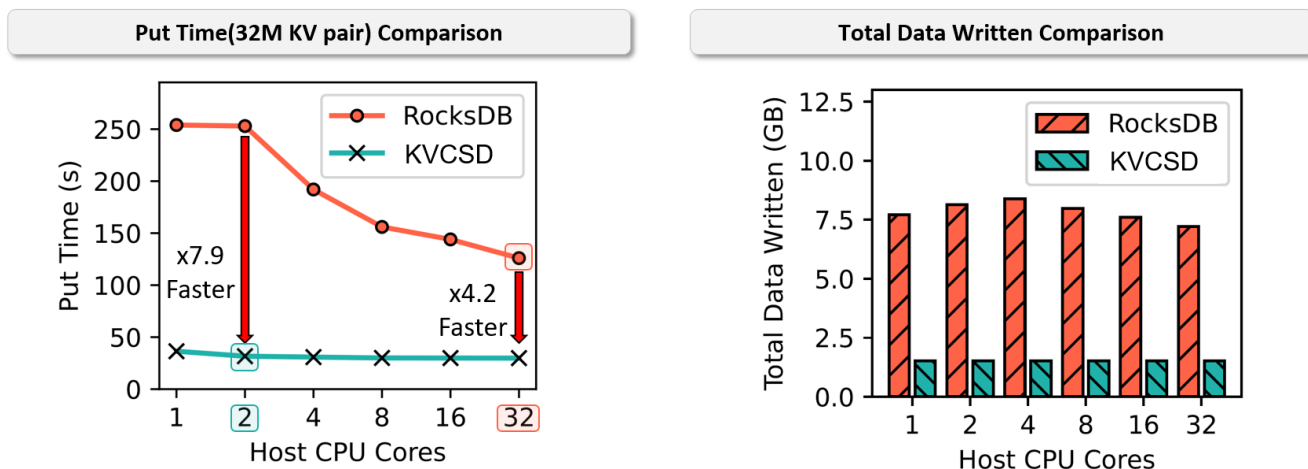


Figure 12 : Higher Write Performance with fewer cores

The scientific community relies heavily on large-scale data analytics to make groundbreaking discoveries. However, as data grows rapidly, data queries become highly selective, and host overhead increases, causing the performance of simulations to degrade. Indexes can accelerate analysis, but host-side indexing further degrades overall simulation performance. Computational storage technology presents a unique opportunity to overcome these challenges. KV-CSD is an in-drive, indexing, query accelerating device that offloads both primary and secondary indexing and query processing to the KV-CSD. This reduces the burden on host compute resources and enables fast, efficient data queries. KV-CSD enriches the analysis using multiple secondary indexes and adopts an ordered data structure with ZNS optimization.

7.1.2 Search In Storage Use Case

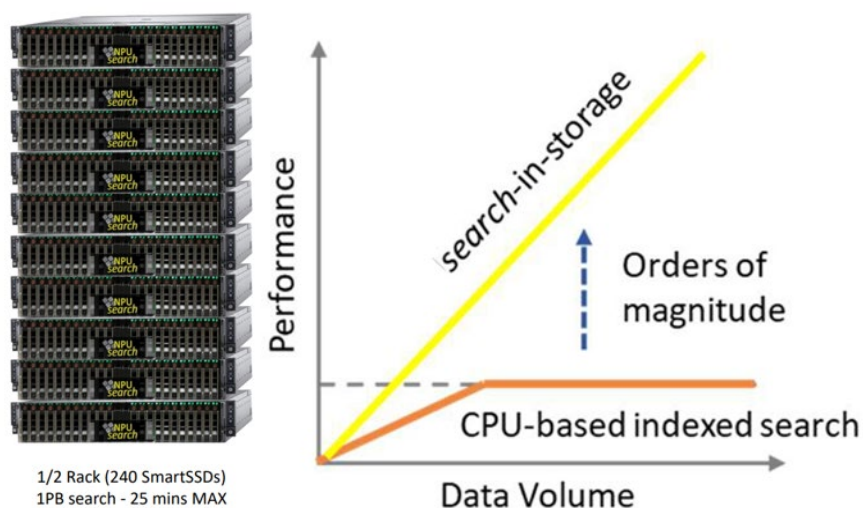


Figure 13: Search-In-Storage – LRL (Lewis Rhodes) Labs

The search in storage case shown in Figure 13 is a SmartSSD²³ which has both an AMD FPGA and a Samsung SSD in U.2 form factor²⁴, is responsible for searching specific logs in the locally stored data. Each sharded data stored in the SSD is processed by the local FPGA independently, so it can achieve high parallel operations.

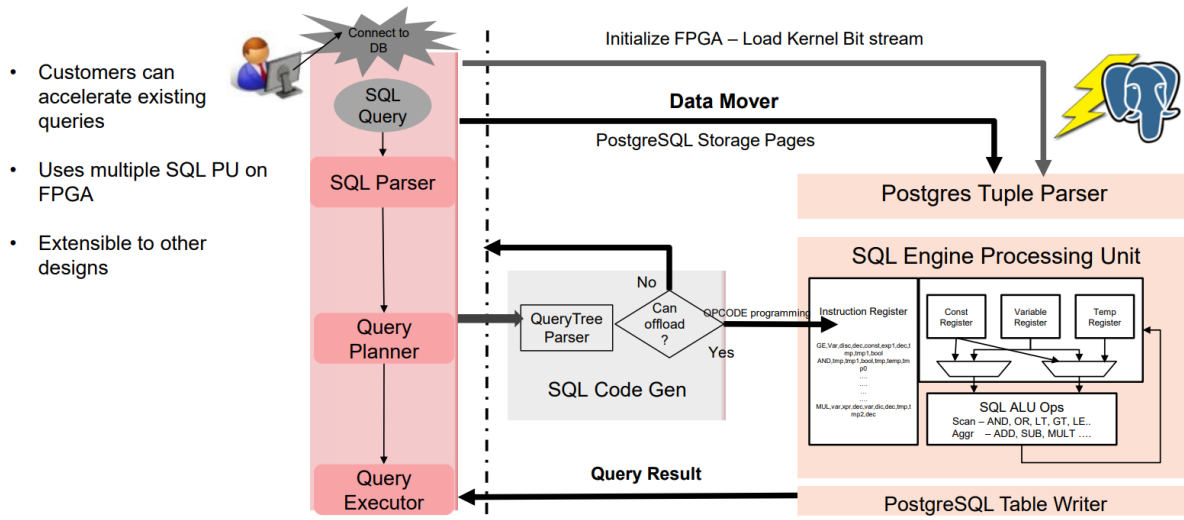


Figure 14: FPGA Accelerated Database

7.1.3 Database Use Case

The database use case is handled by both x86 and the FPGA accelerator. The query planner running on the x86 identifies specific database primitives and offloads specific operations to the accelerator as described in Figure 15. Depending on the query and the workload, this database acceleration can lower a substantial CPU compute cycle and improve end-to-end performance. The most common functions performed by the accelerator are compression, decompression, filter, hash, and data conversion.

²³ SmartSSD <https://semiconductor.samsung.com/ssd/smart-ssd/>

²⁴ <https://semiconductor.samsung.com/news-events/tech-blog/a-giant-leap-forward-in-data-search-is-about-to-change-everything-from-cybersecurity-to-e-commerce/>

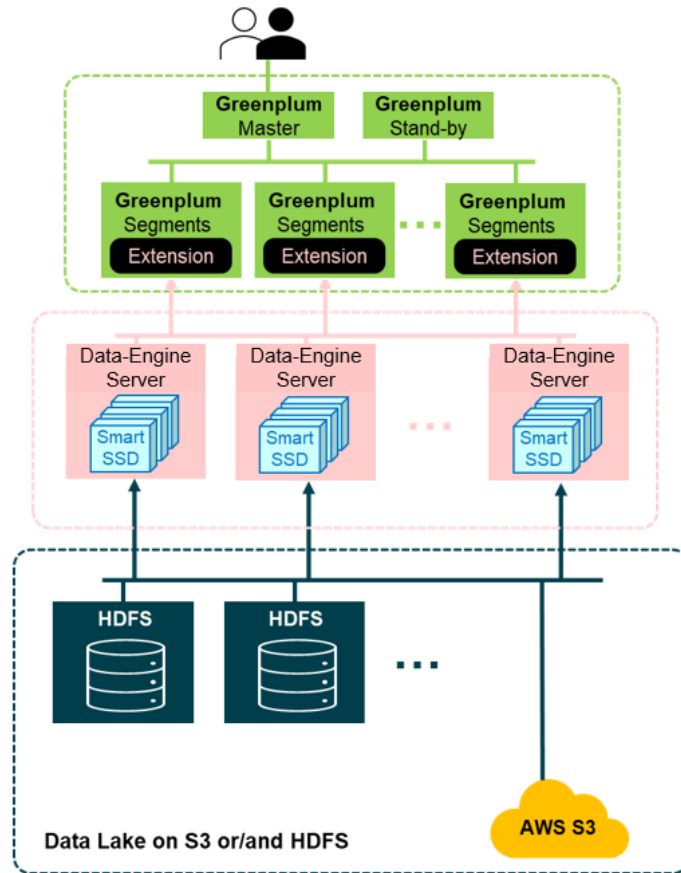


Figure 15 : Greenplum Database^{25 26}

The Greenplum database use case is a strong and unique case of computational storage. Each data engine node introduced in between the Greenplum cluster and the Data Lake²⁷ has a few CSDs (e.g., SmartSSDs). This data engine node enables near data-compute. Specifically, the caching capability in the data engine node improves the time to load the data from the data lake, and additional decompress and scan operations can be done in the FPGA to offload host CPU cycles. The FPGA is a field programmable device, so it can support various acceleration functions as the application requires or changes dynamically.

²⁵ AWS S3: Amazon Simple Storage Service (S3) <https://aws.amazon.com/s3/>

²⁶ HDFS: Hadoop File System https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html

²⁷ Data Lake: Centralized repository for the vast amount of raw data in its native format

7.1.4 Video Analytics Use Case

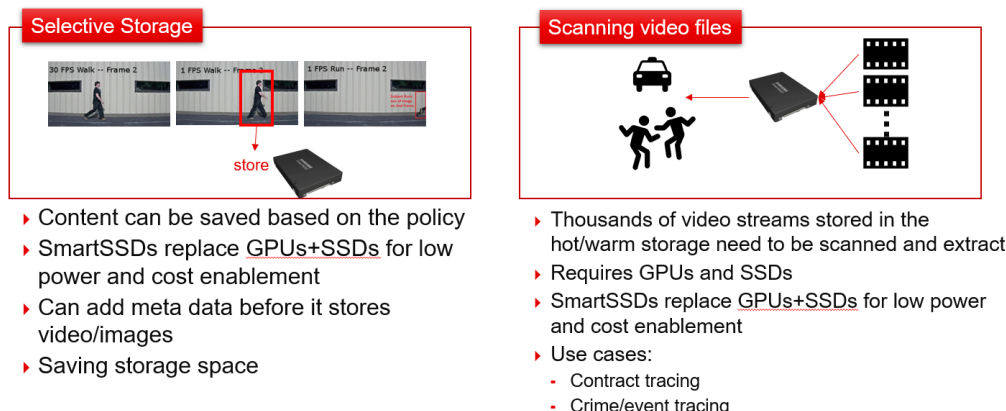


Figure 16 : SmartSSD Video Analytic Use Case

The two video analytics use cases presented in Figure 16 (i.e., Selective Storage and Scanning video files) combine a machine learning function, a video processing function, and a storage function. The Selective Storage use case saves video clips or images based on a policy (e.g., based on the number of people in the frame or whether a specific person is in the frame, save the video clips). It also adds metadata to each video clip or image, so it can be retrieved later without reviewing the entire video file. The Scanning video files use case retrieves raw video files stored in multiple SSDs and processes them in parallel. By running scanning operations on different files in parallel, it can achieve much higher system level performance. In both cases, we can use video decoding and object recognition function in the accelerator in the SmartSSD.

7.1.5 Video Processing Use Case

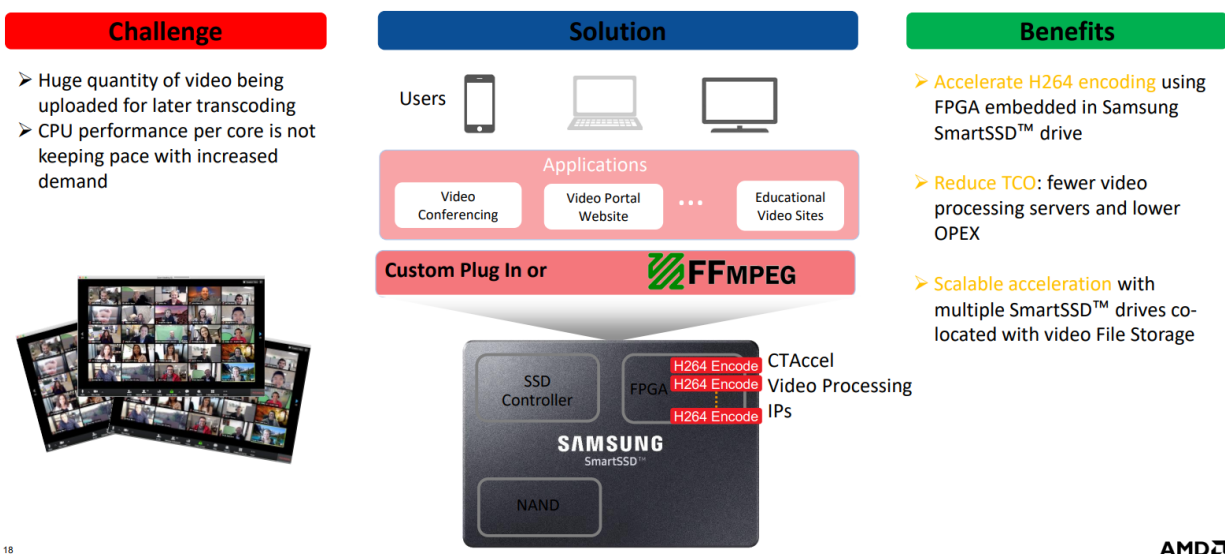


Figure 17 : Video Processing Acceleration for Playback

Figure 17 shows a video processing use case using a CSD (e.g., SmartSSD) utilizing programmable logic space in the FPGA of the CSD. Because of the field-programmable capability, the FPGA can be programmed with only the required video codec or the video codec may be upgraded to a newer version easily.

7.1.6 Data Integrity Check

Checking the integrity of stored data is a common activity in enterprise and data center storage. These checks enable customers to verify data integrity to avoid unexpected data corruption within the storage device. Historically, these CRC data integrity checks were completed on the host. The host computes a CRC for each file and stores the CRC as metadata as files are written. Later, the host decides to check the integrity. The host reads each file as well as the metadata CRC. The host then computes the CRC on the file and compares with the CRC that was read. In other words, the host does all of the work.

Utilizing Computational Storage, the CRC check can be completed on the drive, instead of transferring the data to the host. It is only necessary to transfer the comparison result to the host, and that is what matters to the host anyway. With a CRC check executing in Computational Storage, the host is offloaded, PCIe[®]/NVMe[®] traffic is reduced, the host memory footprint is reduced, and performance scales as drives are added.

Figure 18 shows a block diagram of server on the left and a CSD on the right. In this example, there is an object storage software layer running on the host. The object storage layer breaks the data into shards. Each shard requires one or more LBAs to store the corresponding data. The CRC is computed on each shard as the data is written. When the host decides to perform a data integrity check, the object storage provides the list of shards stored on a particular CSD. The filesystem layer translates each shard into a list of LBAs that comprise that shard and that list is provided to the CSD. Looking at the CSD on the right hand side of the diagram, numbered steps 1 through 6 detail the flow of commands. To summarize, an LBA to check and an LBA with the stored CRC are copied from NAND to an in-drive buffer. A CRC engine computes the CRC on the LBA and compares with the stored CRC. The result of the comparison is retrieved by the host. The CRC calculation can be performed on sharded data as long as the stored CRC aligns with the shard on that drive.

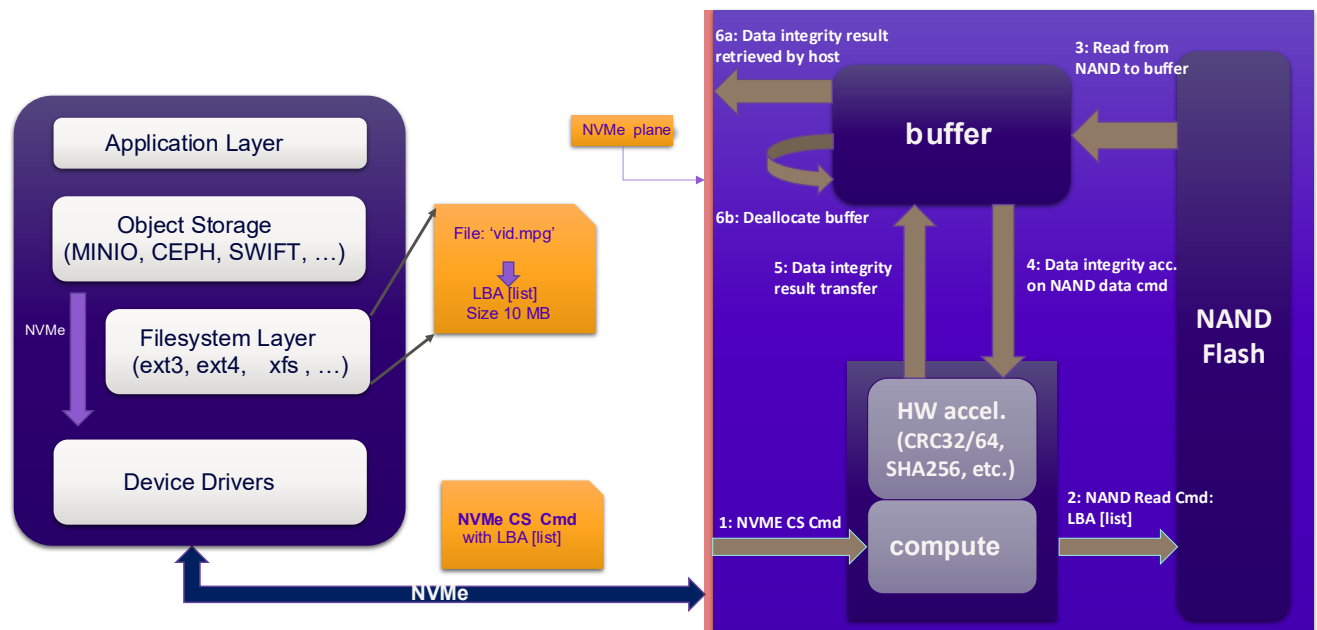


Figure 18 : Data Integrity Check CSD

Figure 19 shows that performance continues to increase as drives are added (orange bars) while the host performance to compute the CRC (blue bars) levels out as the bus is saturated moving the data. Clearly, performing the CRC calculations in parallel in each drive will complete the operation faster while saving both host and PCIe bandwidth.

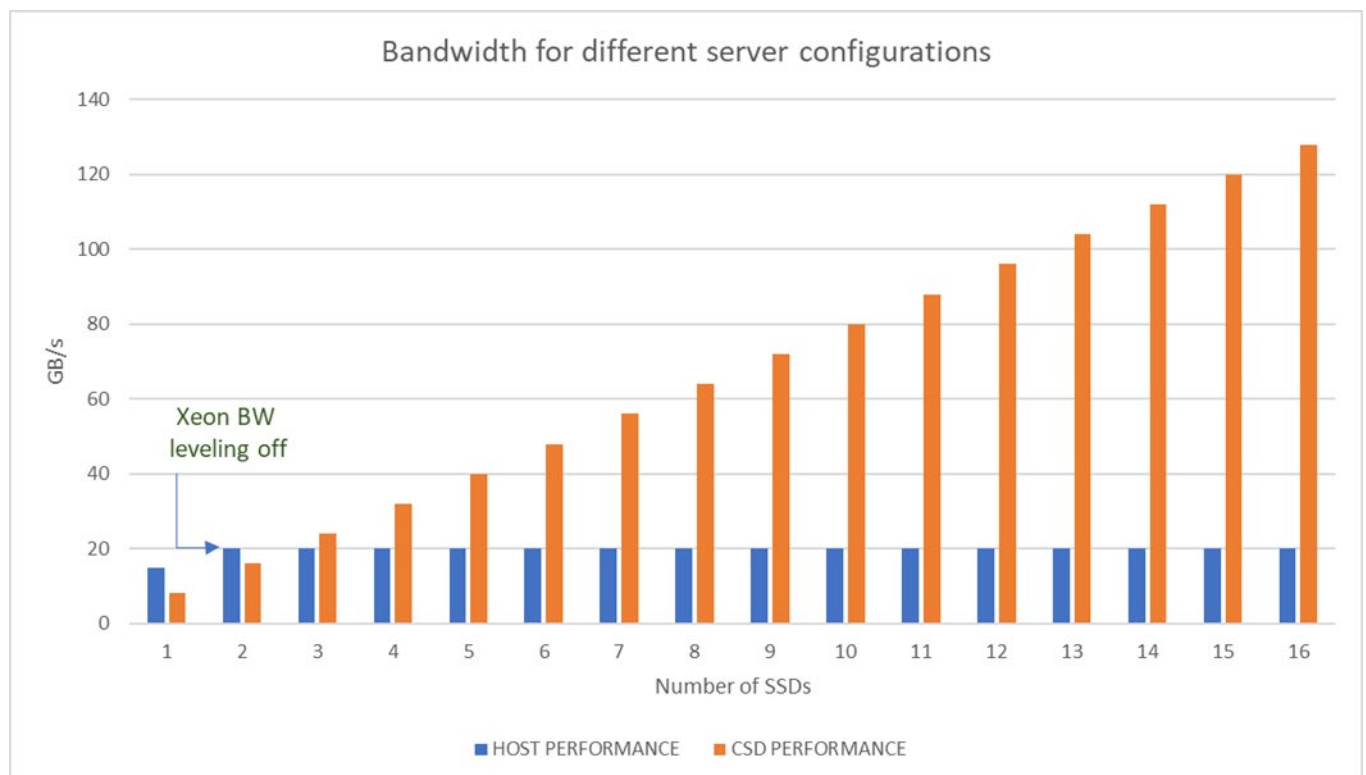


Figure 19 : CSD Performance Scaling

In summary, a CRC data integrity check executing on a CSD is a great example of NDC. Moving the compute to the data saves host bandwidth, reduces PCIe bandwidth, and scales as drives are added. Sharded data is not a problem, which is a huge benefit to customer concerns about data locality.

7.2 Processing-In-Memory Use Cases

7.2.1 One Memory-Coupled Computing (OneMCC)

In the rapidly changing world of computing, the pursuit of memory-coupled computing has led to innovative methods. OneMCC²⁸ for The Open Compute Project is an integrated software stack for memory-coupled computing. OneMCC is a transformative ecosystem that includes a range of advanced technologies, from PIM to PNM technologies. This framework integrates programming models, compilers, runtimes, acceleration libraries, and user/kernel-level drivers, providing a comprehensive solution to unlock the full potential of memory-coupled computing.

OneMCC is designed to be generally used for various domain applications, including High Performance Computing (HPC), databases, Operating System level system functions, and AI domains such as Deep Learning (DL). To support offloading computation to the MCC device according to domain-specific applications, OneMCC provides an API and programming models. The MCC device may be configured as a PIM device, a PNM device or a PIM-PNM device connected via a Compute Express Link™ (CXL™) and other interfaces.

Using OneMCC, an NDC device is programmed for computation as follows:

- The PE loads data from the memory bank into appropriate register files according to the implementations.
- The PE executes NDC instructions with the information of the memory addresses of the operands given by the memory command.
- Upon completion of the NDC instruction, the PE stores data from the register file into memory.

Figure 20 illustrates the execution model of a PIM device. The PIM architecture is designed to enable a drop-in-replacement of current JEDEC²⁹ compliant DRAM, and hence, may be controlled by the standard memory commands from the memory controller. Accordingly, the PIM execution model is designed to transparently scale the bank-level parallelism of the PIM via the memory commands. Therefore, the instructions may be in the main memory of the host system or in the memory of the host device which is connected to the host via the hardware interface (e.g., CXL, PCIe). The memory controller generates the memory command by scheduling the memory request from the PIM kernels. The PIM kernel is a program that generates memory requests to the PIM device with the correct addresses to fully utilize the on-chip compute bandwidth of the PIM device. Alternately, a PIM binary consists of a set of PIM instructions and resides in the instruction buffer within the PIM device.

²⁸ OneMCC (Memory Coupled Computing) infrastructure <https://github.com/SAITPublic/OneMCC>

²⁹ Global standards for the microelectronics industry
<https://www.jedec.org/category/technology-focus-area/main-memory-ddr-sdram>

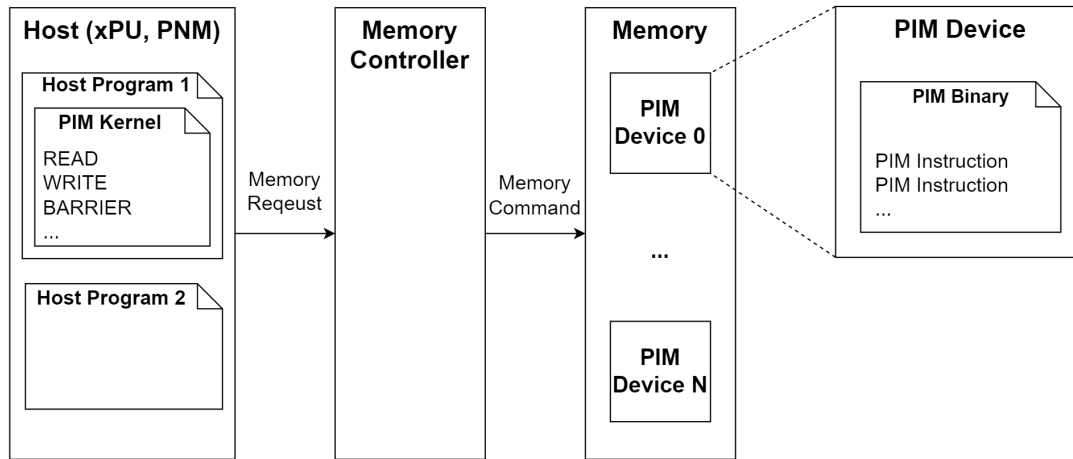


Figure 20 : Execution model of the PIM device

The PIM kernel may be written using a host programming model (e.g., OpenCL (see 4.3.1.3)), HIP³⁰, or CUDA³¹). The host processor may be either a processor, a CXL controller or a PNM device depending on the hardware design.

Figure 21 illustrates the domain model of OneMCC. In a target domain such as an AI domain, DL application is the main workload and DL framework is generally used to execute the DL application. DL framework has its backend implementation to execute DL application on the heterogeneous hardware system. OneMCC provides support for multiple programming models and backend implementations as shown in Figure 21. General programming model based applications can seamlessly offload their program to the MCC device through OneMCC. This model provides an LLVM³² compiler extension and PIM primitive support by extending the LLVM compiler. OneMCC is able to easily integrate with other general-purpose programs by sharing the compiler stack (i.e., LLVM middle-end and back-end), which generates the PIM kernel.

³⁰ C++ Runtime API and Kernel Language <https://rocm.docs.amd.com/projects/HIP/en/latest/>

³¹ A development environment for creating high-performance, GPU-accelerated applications in NVIDIA <https://developer.nvidia.com/cuda-toolkit>

³² LLVM <https://llvm.org/>

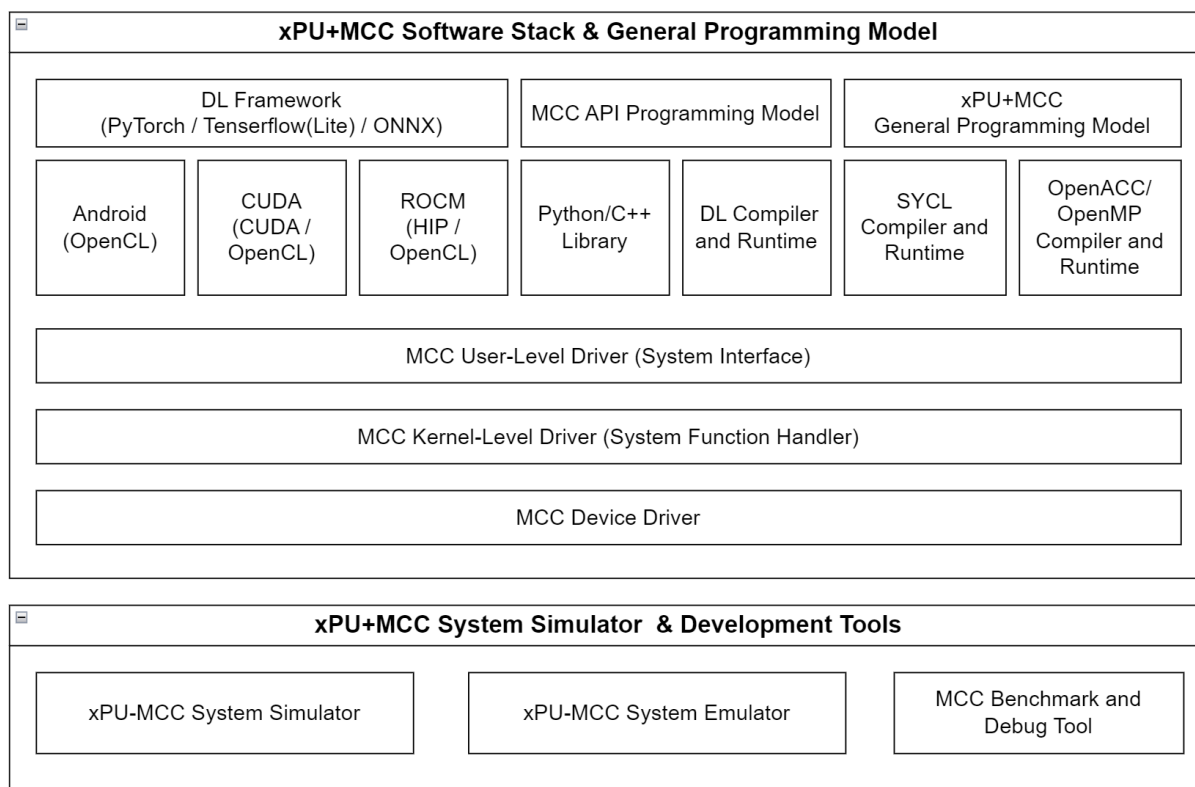


Figure 21 : Domain model of OneMCC

In summary, OneMCC tackles the critical challenge of memory-coupled computing by seamlessly integrating hardware and software within its framework. This innovative approach optimizes deep learning models, empowers general-purpose applications, and redefines computational efficiency. The implications of this knowledge acquisition are profound, setting the stage for a future where memory-coupled computing becomes the new norm, unlocking unprecedented possibilities for performance, energy efficiency, and innovation. OneMCC not only addresses today's computational challenges but also paves the way for a brighter, more efficient, and more capable future in computing.

7.3 Processing-Near-Memory Use Cases

7.3.1 AI Framework Use Case

Having selected the most memory sensitive AI models that would stand to gain the most from executing on a PNM system, a model for test was created as shown in Figure 22, and a generic algorithm was created that would optimize both memory and compute placement.

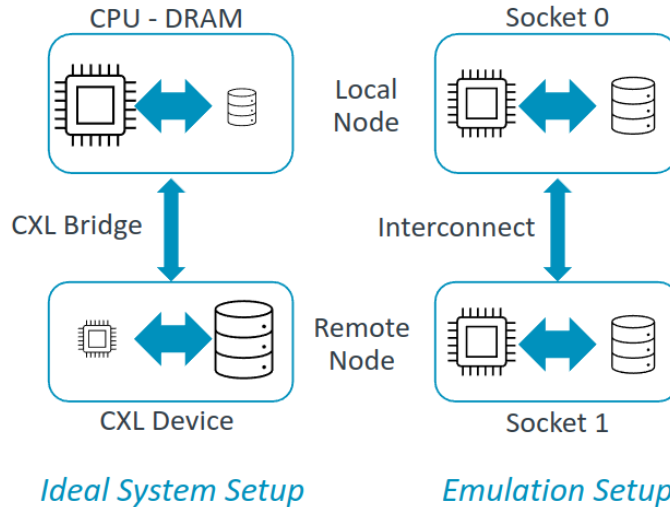


Figure 22 Emulation setup mocking CXL delays with NUMA cross-socket delays

The emulation setup required executing on a multi-socket machine and utilized a modified TensorFlow³³ (see 4.3.1.5) which used libnuma to select whether memory (both static, in the form of weights, and dynamic intermediate layer outputs) should exist in the local or remote domain, and thread affinity to force computation to execute on the local or remote domain. The far socket's clock rate was slowed to emulate the attached device having less computation compared to the host.

Each layer of the model was independently profiled for memory sensitivity and compute sensitivity and that profiling data was stored. The placement algorithm takes as input the model itself, the profiling data for each layer of that model, and a bias toward optimizing either time or space concerns. Because each layer takes as input the output of the previous layer, the algorithm makes multiple passes to optimize its annotations. The output is a fully annotated model which describes where the memory should live between host and on-device, and where the compute should happen between host and near-memory. Figure 23 shows a simple model diagram and that same model with its annotations.

³³ Martín Abadi, et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. <https://www.tensorflow.org/>

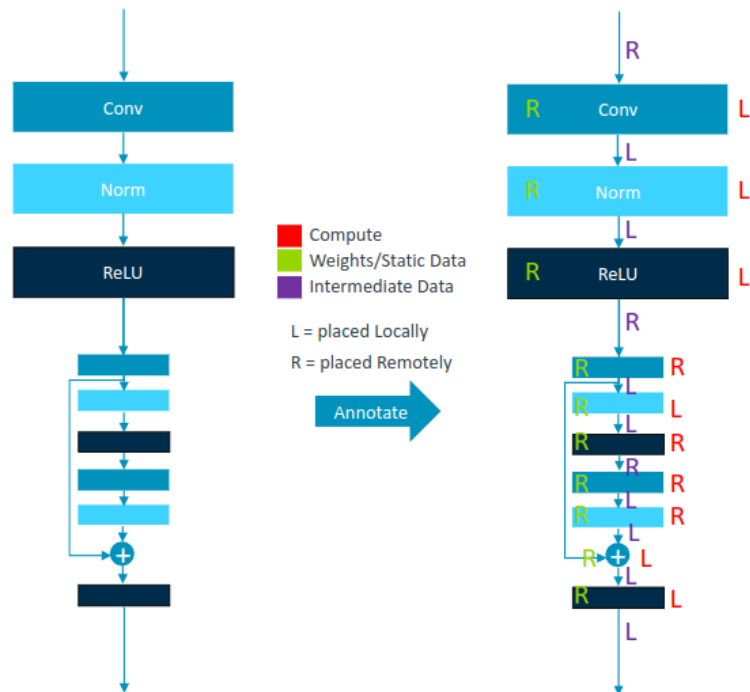


Figure 23 Example AI model with compute/data placement annotations

When these annotations are followed, we observed a strict efficiency improvement over naive designs which either failed to use PNM or failed to use attached memory. Figure 24 shows these results. In the lower-left corner we have a maximal amount of attached memory used with minimal execution time. The placement algorithm (partitioner) produces configurations that are below and to the left of any random or naive configuration – showing that optimal performance requires correct compute- and data-placement.

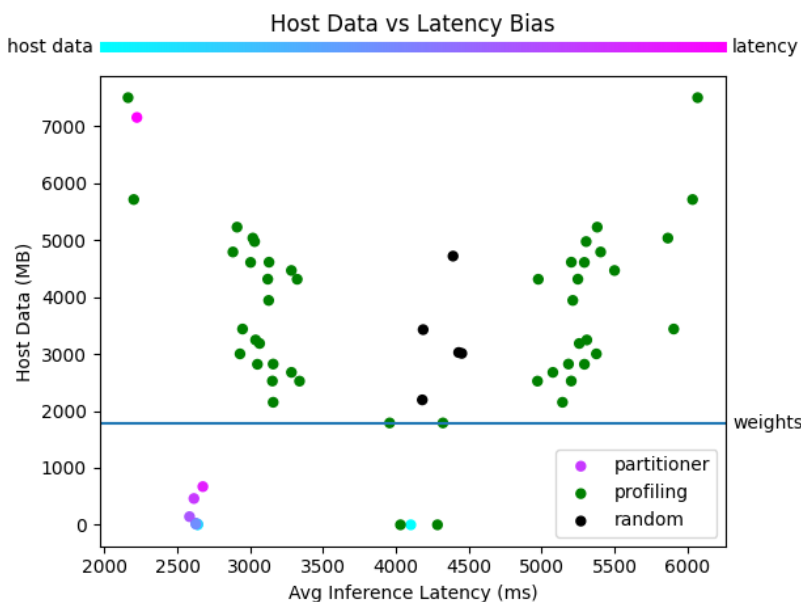


Figure 24 Run results from an array of Stable Diffusion ML model configurations

7.3.2 VectorDB Offload Use Case

Having identified the use cases that would benefit most from PNM, and having identified a process by which total run-time could be optimized while executing on a PNM system by selectively offloading the most critical compute tasks, we found that the software changes needed would not be onerous.

Using a design as in Figure 25, compute was executed in a Linux³⁴ environment. The software was modified from a monolithic application to a client-server co-processing application where the host sends work packets through a client to a server running near memory. This does require the sharing of memory, or at least the sharing of a portion of memory needed for communication between host and device.

The result in Figure 26 shows that performance of an offloaded task near-memory is only slightly slower than the host while fully utilizing remote memory, freeing the host's compute for other tasks.

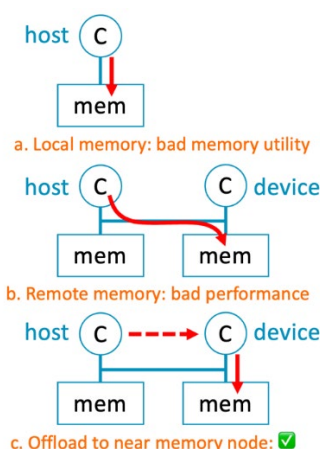


Figure 25 Diagram of various compute offload options

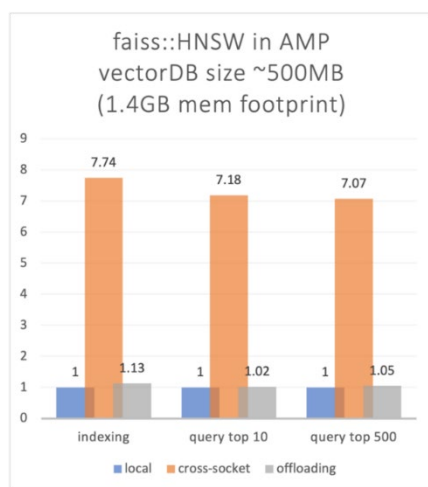


Figure 26 Runtime performance comparisons of an identified memory-sensitive workload, host v. offload

³⁴ Linux Operating System <https://tldp.org/LDP/intro-linux/intro-linux.pdf>

7.4 Data Processing Unit (DPU) Use Cases

7.4.1 DPU Use Case

Software-defined infrastructure runs today's data centers, where VMs, containers, or agents using host CPU for virtualization, networking, storage, security, management, and provisioning. While general-purpose CPUs excel at general-purpose, single-threaded workloads and are typically optimized for performance, they are not known for power efficiency, due to which these infrastructure workloads may consume 30% of the processor cycles. Domain specific accelerators are better at such jobs and tend to improve performance, reduce power consumption, while freeing up CPU cycles for other applications and workloads. A DPU is better tasked at running these infrastructure workloads since they have dedicated hardware engines for accelerating networking, data encryption/decryption, key management, storage virtualization, and other tasks, while being more power efficient when compared to the general purpose CPU.

The NVIDIA Bluefield³⁵ DPU includes domain-specific accelerators for networking, storage, and security, along with energy-efficient Arm CPU cores to offload the control plane of infrastructure applications and/or parts of the hypervisor or container management software. The DPU has direct access to the networking pipeline, and can perform SDN, telemetry, deep packet inspection, or other networking tasks more efficiently than the server CPU.

As part of zero-trust security requirements in data centers, encryption on every network link is increasingly employed, thereby protecting data even when a server on the network is compromised. IPsec is the most popular tool for encrypting data between servers and clients as part of security options. But this places a high cost on CPU usage at high network speeds. A SmartNIC or DPU is more suited for this job where offloading can perform IPsec encryption/decryption faster and with less power compared to the general purpose CPU. Figure 27 shows the power savings when IPsec is offloaded to a DPU. The power savings are significant on both the client and server side.

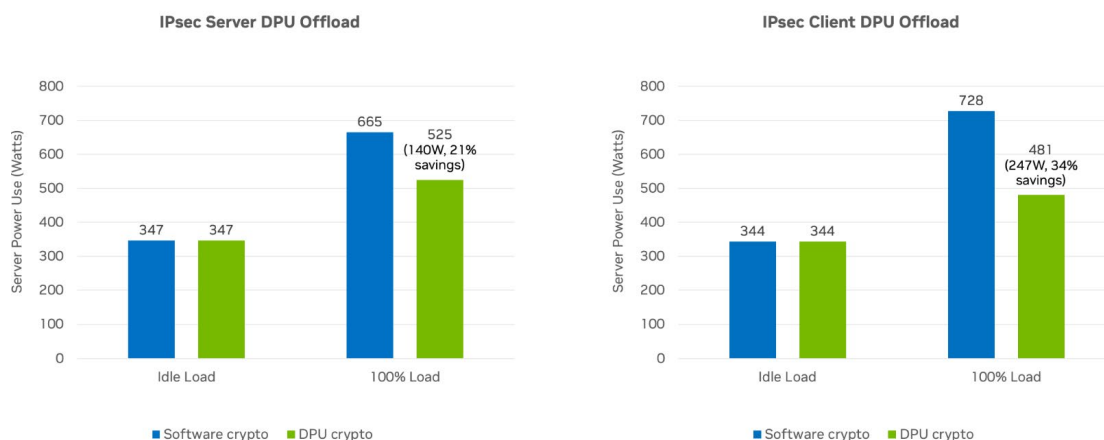


Figure 27: DPU offload power savings

³⁵ DPU Power Efficiency <https://resources.nvidia.com/en-us-sustainable-computing-microsite/en-us-sustainable-computing/nvidia-dpu-power-efficiency-white-paper>

8 Layer Abstractions

8.1 Introduction

NVMe, SNIA, and OCP NDC have been working on developing standards and specifications for computational storage for several years, specifically working to define interfaces and protocols that enable efficient communication between NDC devices and host systems. The goal is to create a standardized and interoperable ecosystem for near-data compute interfaces that can be used across different hardware and software platforms.

Abstraction may be done at different levels. This section will describe abstraction at the end user level and abstraction at the platform level.

8.2 End User Interface

8.2.1 Near Data Computing Interface

Before we define a near-data compute API, let's discuss lower-level components first. Microservices are a software development technique that structures an application as a collection of small, independent services that communicate with each other via APIs. In the context of NDC, microservices can be used to provide a more efficient and scalable way to manage data processing and storage tasks. The software framework stack, on the other hand, is a set of software components that work together to provide a complete solution for a particular task. In the context of NDC, the software framework stack can include various components such as operating systems, file systems, device drivers, and APIs that invoke microservices via various adaptation layer. The exact components of the software framework stack can vary depending on the specific use case and requirements. Figure 28 shows the early data transformation using NDC.

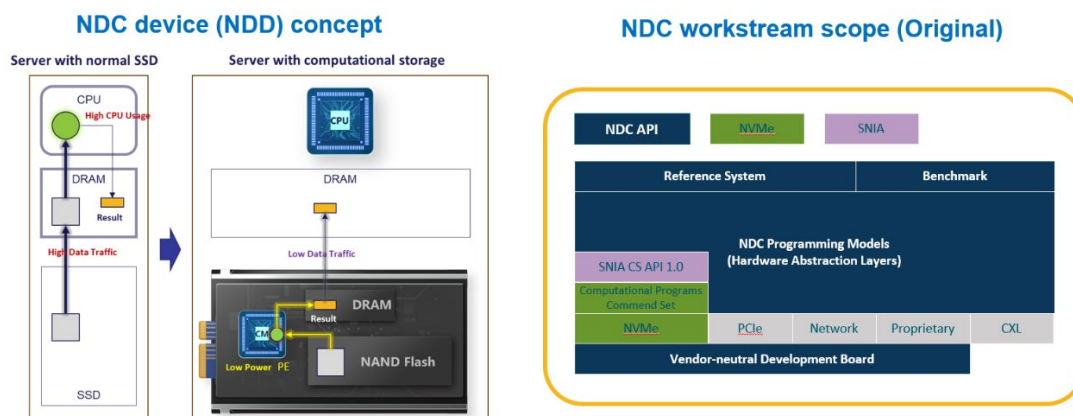


Figure 28 : Early Data Transformation Using Near Data Computing

NDC is a technology that enables data processing and storage to be performed on the same device or data processing done right next to the storage device, thereby reducing data movement and improving performance. The software framework stack for microservices in end user perspective for computational storage includes several protocols.

- Firstly, the NVMe protocol is used to communicate with the computational storage device. The NVMe Computational Programs Command Set³⁶ provides a standard interface for computational storage devices. The SNIA Computational Storage API³⁷ (SNIA CS API) is another specification that provides a transport protocol agnostic standard way for applications to access computational storage devices.
- Secondly, the PCIe³⁸ protocol connects the computational storage device to the host system. PCIe is a high-speed serial bus that provides low latency and high bandwidth.
- Thirdly, the network protocol (e.g., TCP) enables remote access to computational storage devices, which is useful for distributed systems.
- Finally, the CXL³⁹ protocol is a new interconnect standard that provides low latency and high bandwidth. It is designed to enable high-performance computing and is compatible with both PCIe and memory protocols.

The software framework stack for microservices in end user perspective for NDC includes the NVMe, PCIe, network (e.g., TCP), and CXL protocols. These components can work together to enable efficient and high-performance data processing and storage on the same device.

As shown in Figure 28, the NDC programming interface is placed on top of SNIA CS API. This SNIA CS API interfaces with the NVMe Computational Programs Command Set. NDC approach is to connect the application with various other devices as listed in Figure 28 with a flexible and easy to develop interface.

Figure 29 and Figure 30 show a detailed view of the SW Stack Details from NVMe, SNIA, and OCP NDC.

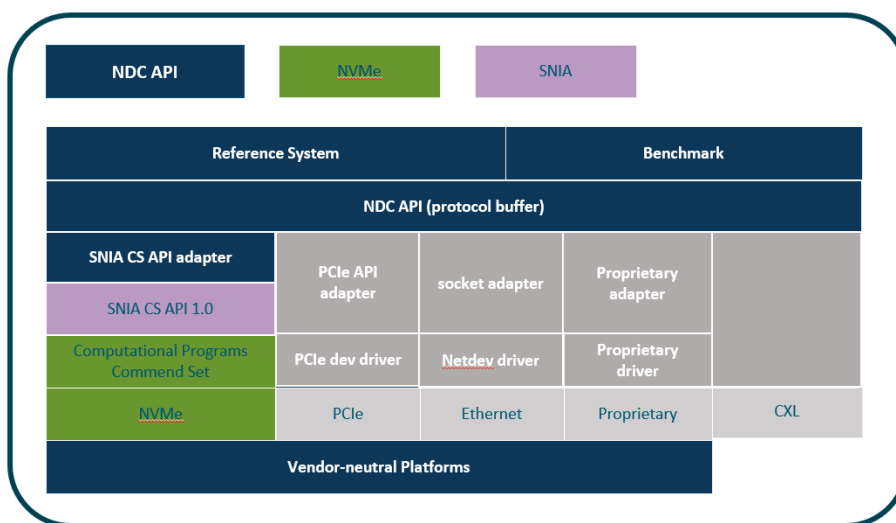


Figure 29 : SW Stack Details from NVMe, SNIA, and OCP NDC CS

³⁶ NVM Express Computational Programs Command Set, <https://nvmexpress.org/wp-content/uploads/NVM-Express-Computational-Programs-Command-Set-Specification-1.0a-2024.07.09-Ratified.pdf>

³⁷ SNIA Computational Storage API, <https://www.snia.org/csapi>

³⁸ PCI Express (PCIe) is a highly scalable, point-to-point interconnect <https://pcisig.com/specifications>

³⁹ Compute Express Link (CXL) is an industry-supported Cache-Coherent interconnect for processors, memory expansion and accelerators <https://computeexpresslink.org/>

As shown in the example stack diagram in Figure 29, OCP NDC may need to develop different adapter-level APIs, like SNIA CS API for other devices. However, this is very challenging as each hardware accelerator requires its own adaptation layer. So the following block diagram proposes a new idea to reuse SNIA CS API for all computational devices.

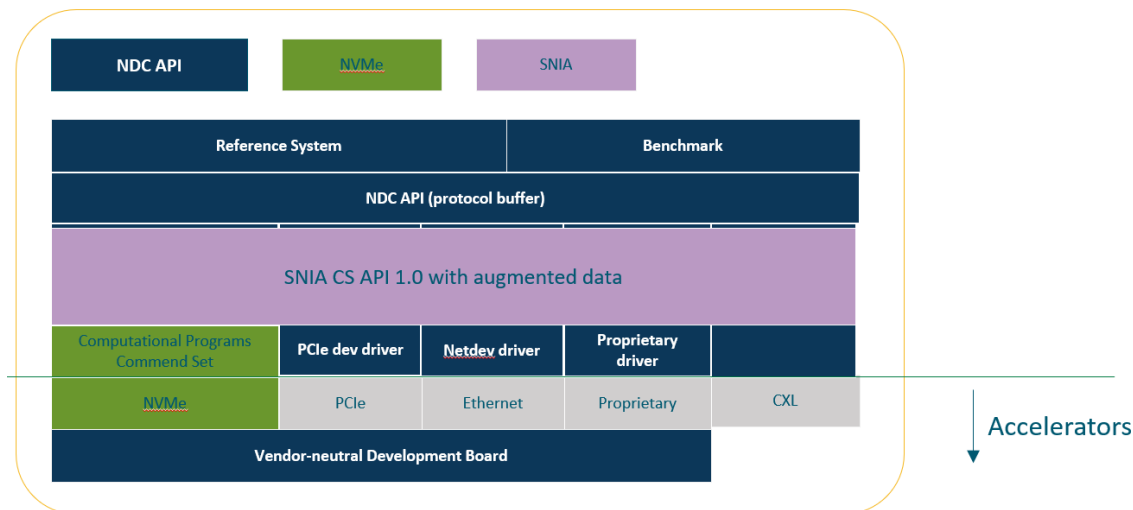


Figure 30 : SW Stack Details from NVMe, SNIA, and OCP NDC

One way to enable a unified interface is to utilize the SNIA CS API interface, The API needs to bring advanced parameters specific to each device out as API arguments while standard arguments are carried in the SNIA CS API functions.

Depending on the function, where it might have application specific parameters, the user can specify them through a JSON type file. So:

- Protocol buffer based API generates serialized data messages which include command, parameter and data
- Serialized data is transmitted through PCIe/Ethernet/Proprietary interface
- The agent in the NDC device deserializes the data of the message and issues the command with parameter to NDC Engine (NDCE)

The block diagram in Figure 31 shows an example API structure. The standard SNIA CS API carries the name of CS devices, host input and output memory locations and others. However, devices with advanced functions require more parameters to control detailed operations. These information can be carried as a part of the data.

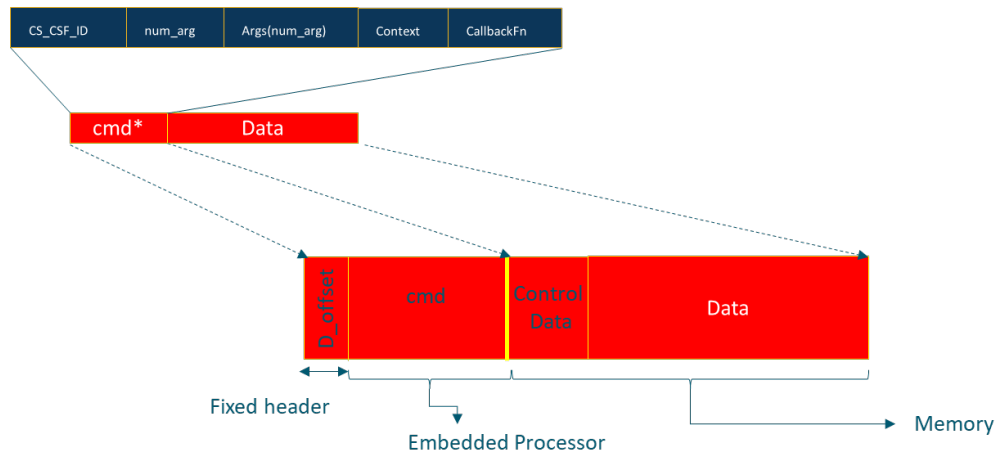


Figure 31: NDC API carrying device specific control and data

The assumption in Figure 31 is that the NDC device has a small processor to parse the arguments and generate device specific commands.

8.2.1.1 NDC API

The NDC API is a new API that aims to simplify the use of the SNIA API, which is designed for computational storage devices. The SNIA API is similar to the OpenCL API, which is used for more general platforms like accelerator boards. The NDC API encapsulates the SNIA APIs used in a specific use case, such as the H.264 video encoder (see 4.3.2.5), by bringing out the necessary generic parameters as API arguments while leaving anything that can be localized inside. Depending on the CS function, where it might have application-specific parameters, users can specify them through a JSON file (See Figure 32). The host will parse this API and convert it to the SNIA CS API.

► Message file should include:

- type ('open')
- encoding ('h264', 'raw', 'jpeg')
- width
- height
- network (name of ML network on VMSS server)
- protocol (rtp | rtsp)
- client IP
- camera URL (if rtsp)

```
{
  "type": "open",
  "encoding": "h264",
  "width": 224,
  "height": 224,
  "network": "googlenet",
  "protocol": "rtp",
  "clientIP": "127.0.0.1"
}
```

```
#include "CS_client.h"

ftiMessage open_session_msg;
fti_client_new_msg_from_json("./json_msg_file.txt", &open_session_msg);
```

Figure 32 : Example Message Object in JSON

8.3 Platform Developer Perspective

8.3.1 Programming Options

As a platform developer, there are many choices available in programming NDC such as OneAPI, ROCm, OpenCL, RUST, Python, SNIA CS API, and Substrait.

8.3.1.1 oneAPI

Originally developed by Intel and now maintained by Unified Acceleration Foundation ((UXL), oneAPI⁴⁰ is a unified programming model developed to address the challenges of heterogeneous computing environments, where multiple types of processors and accelerators may be used in tandem to support diverse workloads. This approach allows developers to write code once and deploy it across multiple hardware architectures, including CPUs, GPUs, FPGAs, and AI accelerators. By facilitating code portability and reducing the need for multiple specialized codebases, oneAPI is positioned to streamline development workflows, potentially lowering development costs and time-to-market while maintaining high performance.

OneAPI is built upon open standards, notably SYCL⁴¹ with Modern C++ and Data Parallel C++ (DPC++)⁴², with high-performance libraries⁴³ that include a range of performance-optimized libraries, analysis tools, and debugging tools⁴⁴. Samsung's memory-centric computing for PIM and PNM technologies uses extensions to oneAPI to maximize parallelism and power efficiency (see section 3.2.1).

While CUDA remains widely adopted in AI and machine learning and primarily supports NVIDIA GPUs, oneAPI provides a multi-architecture alternative for organizations looking to avoid vendor lock-in. There are also tools to simplify migration and aid in porting CUDA code to DPC++.

Even with this range of benefits, oneAPI does have potential challenges and limitations such as, the learning curve with new standards (i.e., SYCL or DPC++), time to transition from established frameworks like CUDA, variable performance with a variety of hardware, ecosystem maturity, and industry adoption.

8.3.1.2 ROCm

The AMD ROCm^{TM45} (Radeon Open Compute platform) open-source software platform is optimized to extract the best HPC and AI workload performance from AMD InstinctTM GPU accelerators while maintaining compatibility with industry software frameworks. ROCm consists of a collection of drivers, development tools, and APIs that enable GPU programming from low-level kernel to end-user applications and is customizable to meet user specific needs. Being open-source, ROCm helps to develop, collaborate, test, and deploy applications. Once designed, the software is then portable, allowing movement between accelerators from different vendors or between different inter-GPU connectivity architectures - all in a device-independent manner.

⁴⁰ oneAPI from Unified Acceleration Foundation ((UXL) <https://oneapi.io/>

⁴¹ SYCL is a C++-based, cross-platform abstraction layer from the Khronos Group

⁴² DPC++ is the primary programming language for oneAPI, which extends SYCL with additional features to improve programming efficiency, such as unified shared memory and sub-groups

⁴³ Data Parallel C++, Deep Neural Network, Collective Communications, Data Analytics, Threading building blocks and Math Kernel libraries

⁴⁴ Intel VTune profiler for performance analysis <https://www.intel.com/content/www/us/en/docs/vtune-profiler/user-guide/2025-0/overview.html>

⁴⁵ AMD ROCm Open Software for AI and HPC Workloads <https://www.amd.com/en/products/software/rocm.html>

ROCm is powered by Heterogeneous-computing Interface for Portability (HIP). It supports programming models, such as OpenMP and OpenCL, and includes all necessary open-source software compilers, debuggers, and libraries. It's fully integrated into machine learning (ML) frameworks, such as PyTorch⁴⁶ and TensorFlow. ROCm consists of machine learning and computer vision libraries, communication libraries, math libraries and tools. Specifically, math libraries, such as hipBLAS⁴⁷ or hipRAND⁴⁸, can be used for near data compute accelerations.

8.3.1.3 OpenCL

Open Computing Language (OpenCL)⁴⁹ is a royalty-free open standard for cross-platform parallel programming developed to support a broad range of heterogeneous computing environments such as CPUs, GPUs, FPGAs, and accelerators from different vendors. Originally developed by Apple and now maintained by the Khronos Group, OpenCL enables developers to write portable that preserves the option to switch or upgrade components without extensive code rewrites.

Based on C99⁵⁰, OpenCL includes an API for defining kernels (i.e., the core computational units), a runtime system to manage execution across different devices, and a programming model, for writing kernels designed to scale across multiple cores by taking advantage of the parallelism that can be executed across various types of hardware. OpenCL has a large library ecosystem such as OpenCV⁵¹ for computer vision and TensorFlow for machine learning and extensive documentation.

However, OpenCL has faced competition from both proprietary solutions (e.g., CUDA), and newer standards (e.g., SYCL, which seek to modernize parallel programming with C++ compatibility). OpenCL's API requires explicit management of resources (e.g., memory and parallel execution) which may require additional development time compared to higher-level abstractions and hardware optimizations may still be necessary to maximize performance on specific devices. Additional challenges (e.g., performance variability across hardware, lower-level API complexity, slower evolution, community and ecosystem limitations, and evolving competing standards) may slow down future adoption.

8.3.1.4 RUST

Rust⁵² is a modern, general-purpose programming language designed for performance, reliability, and productivity. It emphasizes memory safety, concurrency, and efficient performance without a garbage collector. Rust is widely used in systems programming, web development, and embedded systems, and is known for its strong type system and ownership model that prevent common bugs at compile-time. This makes Rust an ideal choice for developing performance-critical applications, such as operating systems, game engines, and web browsers. Rust's emphasis on concurrency and safety also helps reduce development time and costs by catching errors early in the development process. Rust introduces several technical innovations such as ensuring memory safety by enforcing rules about how memory is allocated and deallocated, compile-time checkers to ensure valid references and preventing data races, abstractions that do not incur runtime overhead. Rust has gained significant popularity and adoption in recent years with a strong ecosystem of libraries and tools, community development, new use cases (e.g., WebAssembly, embedded systems, and high-performance computing), and efforts to standardize and integrate it with other technologies. Rust is used by major companies like Mozilla, Dropbox, Google, and Microsoft. It has also been adopted for

⁴⁶ PyTorch is an optimized tensor library for deep learning using GPUs and CPUs <https://pytorch.org/index.html>

⁴⁷ hipBLAS is a Basic Linear Algebra Subprograms (BLAS) marshalling library <https://github.com/rocm/hipBLAS>

⁴⁸ hipRAND is a RAND marshalling library <https://github.com/ROCm/hipRAND>

⁴⁹ OpenCL <https://www.khronos.org/opencl/>

⁵⁰ C programming language standards https://iso-9899.info/wiki/The_Standard

⁵¹ OpenCV is a real-time optimized Open Computer Vision Library <https://opencv.org/>

⁵² Rust programming language <https://www.rust-lang.org/>

the development of the Linux kernel, making it the first language other than C and assembly to be supported.

Rust does come with its set of challenges and drawbacks such as learning curve, complexity with its strict type system and ownership model, longer compilation time, limited ecosystem (compared to Python or JavaScript), and tooling maturity. There are also several alternatives to Rust such as C, C++, Go (Golang), Nim and Swift that come with their own strengths and weaknesses.

With its built-in concurrency feature, memory safety and performance, Rust is efficient for parallel processing large-scale data pipelines with low overhead, making it most suitable for near data processing in the near future.

8.3.1.5 Python

As a top ranking programming language⁵³, Python is widely used across various domains, including data science, artificial intelligence, scientific computing, web development, and automation. It is a high-level, interpreted programming language known for its simplicity, readability, versatility and rapid prototyping, making it a powerful tool for both beginners and experienced developers. Python's technical innovations include dynamic type system, interpretive nature, extensive standard library for many common programming tasks that reduces the need for additional code, and robust ecosystem of third-party libraries like NumPy⁵⁴, pandas⁵⁵, TensorFlow, PyTorch, and Matplotlib⁵⁶ extend its capabilities, making it suitable for everything from numerical computation to web development.

Python may be limited by performance execution times due to its interpreted nature compared to compiled languages, concurrency issues due to global interpreter lock limiting effectiveness of multi-threading applications and runtime errors that may be hard to debug due to dynamic typing. But Python's popularity is expected to continue growing, driven by its use in data science, AI, and education and related continuous ecosystem expansion. There are ongoing efforts to improve its performance, to make it more competitive with faster languages, and new applications in areas like IoT, embedded systems, and quantum computing, further enhancing its relevance.

As Python continues to evolve and grow, it will remain a key player in the programming landscape, driving innovation and development across various industries. Python influences near data computing through its ease of programming, rapid prototyping capability and rich set of libraries.

8.3.1.6 SNIA CS API

The SNIA Computational Storage API (CS API) is a set of standardized interfaces to interact with computational storage devices (e.g., CSDs and CSAs) and integrate processing capabilities directly into storage systems. The key objectives include a hardware abstraction that simplifies interactions with software, a consistent interface for various computational storage devices, ensuring ease of integration and use, and enhanced data processing to facilitate direct access and manipulation of stored data. The SNIA CS API is designed to enable applications to leverage the full potential of computational storage technologies, improving performance and enabling new use cases in data-intensive environments. The SNIA CS API is a standardized scalable interface that enables direct access to computational storage capabilities, increasing efficiency in data processing tasks and their cost-effectiveness; and interoperability between products from different vendors while handling growing data volumes and varying computational workloads. The functions defined in the SNIA CS

⁵³ The Top Programming Languages 2024, IEEE Spectrum <https://spectrum.ieee.org/top-programming-languages-2024>

⁵⁴ Harris, C.R., Millman, K.J., van der Walt, S.J. et al. Array programming with NumPy. Nature 585, 357–362 (2020). DOI: 10.1038/s41586-020-2649-2. <https://www.nature.com/articles/s41586-020-2649-2>

⁵⁵ pandas is an easy to use open source data analysis and manipulation tool Python library <https://pandas.pydata.org/>

⁵⁶ Matplotlib is a graph plotting visualization library for Python <https://matplotlib.org/>

API are designed to be agnostic to specific device interface protocols, allowing them to support variety of technologies like CXL and NVMe-oF. Additionally, performance, power efficiency and efficient use of resources by offloading data-intensive processing tasks near data helps reduce the load CPUs and network resources while being adaptable to flexible deployment environments.

The SNIA CS API is currently in an active development phase, with the latest revisions of the SNIA Computational Storage Architecture and Programming Model and the SNIA CS API being refined and enhanced. The SNIA standards are addressing some of the challenges and drawbacks such as complexity that may require specialized knowledge and skills, hardware dependency, potential security risks due to direct storage device access, standardization gaps when integrating with other systems and technologies, efficient resource allocation that requires careful planning and optimization, cost of implementation, scaling with growing data volumes and computational demands, and any vendor lock-in limiting flexibility and potentially increasing costs in the long run. The SNIA CS API represents a significant advancement in the field of data storage and processing by providing standardized interfaces for computational storage devices.

8.3.1.7 Substrait

Substrait⁵⁷ is an open source cross-language specification for describing compute operations on structured data. It explicitly declares a set of common operations, their semantics, and their behavior. While the goal is similar to the Apache Arrow⁵⁸ project, where, Arrow is focused on a standardized memory representation of columnar data, Substrait defines what should be done with the data.

Substrait allows users to generalize the compute operations, therefore allowing to disaggregate such operations across the analytics stack. Figure 33 shows an example of this workflow. A client system can create a Substrait plan based on input SQL query and pass it to a query engine. The query engine can then parse the plan into two plans: (1) a pushdown plan and (2) a remainder plan. Pushdown plan consists of a portion of computations that can be performed by the backend storage system, while the remainder plan is executed by the query engine over the filtered data retrieved from storage system. Overall, by disaggregating compute operations, the performance of data analytics stack can be improved as it reduces the computation burden on the query engine and reduces the amount of data transmitted across the stack.

⁵⁷ Substrait - Cross-Language Serialization for Relational Algebra <https://substrait.io/>

⁵⁸ Apache Arrow <https://arrow.apache.org/>

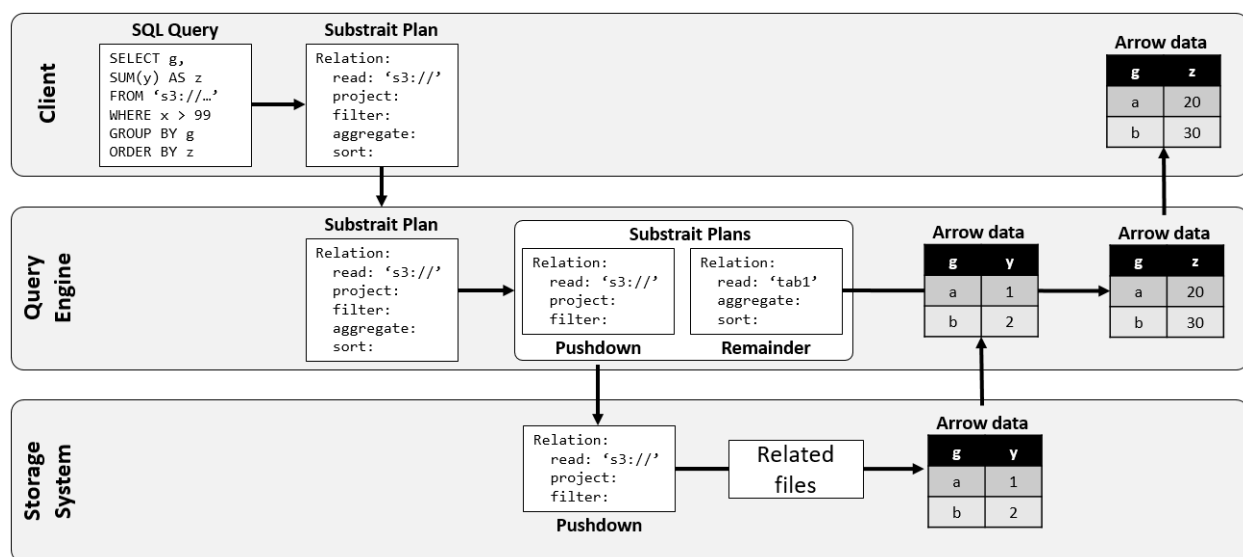


Figure 33: Subtrait Workflow in Data Analytics

8.3.2 Platform Use Cases

8.3.2.1 Object Storage Interfaces

Object storage interfaces ⁵⁹ are gaining popularity in big-data analytics systems, as they offer features such as scalability and unstructured data management. In this section we describe a use case for Computational Storage that leverages an object storage interface to support query pushdown and further accelerate data analytics systems.

8.3.2.2 Object Interface

Object interface is a storage architecture designed to manage large amounts of unstructured data. Object interface stores data in large chunk units (or objects) and appends metadata and a unique identifier so as to locate and access object data. This feature of the object interface can be leveraged by a Computational Storage platform to perform data-aware computation. As shown in Figure 34, in a traditional block interface, data is stored in blocks without any data format. Therefore, block-based computational storage requires additional information (e.g., LBA, range, etc.) to identify related data blocks and perform computation, making them suitable only for data-agnostic operations. However, object interface computation storage can leverage the unique identifier and metadata information attached to each object so as to perform data-aware computation (e.g., filter, sort, etc.).

⁵⁹ Mesnier, Mike, Gregory R. Ganger, and Erik Riedel. "Object-based storage." *IEEE Communications Magazine* 41.8 (2003): 84-90.

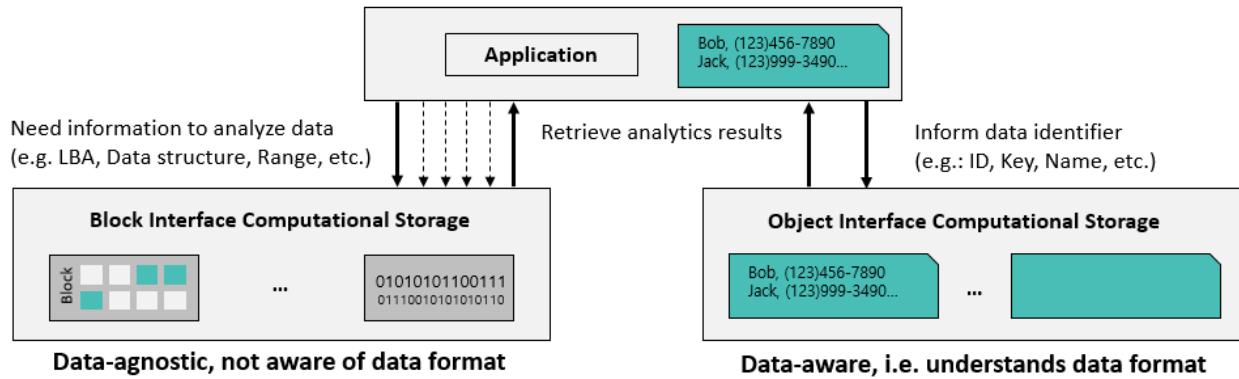


Figure 34 : Comparison of Data-aware and Data-agnostic Storage Interface

8.3.2.3 Query Pushdown with Substrait and Apache Arrow

Cloud storage systems such as Amazon's S3⁶⁰ leverage object interface as it offers scalability and performance benefits. In addition, the S3 API offers a Query Pushdown⁶¹ feature that enables query engines to offload data filtering operations to the backend storage system resulting in reduced data transmission and computation by the query engine. However, the existing Query Pushdown features are limited to a few analytics operations and specific storage APIs. This limits the analytics applications to switch query engines and/or storage systems. One way to address this issue is by employing Substrait to transmit query plans and Apache Arrow to transmit query results.

Substrait is an open source, well-defined cross-language specification describing data compute operations, while Apache Arrow is a language-agnostic, standardized column-oriented memory format to represent data for analytics operations. As shown in Figure 35, Substrait and Apache Arrow provide a seamless interface across various layers and eliminate the burden of selecting desired interfaces to connect to various query engines and storage systems.

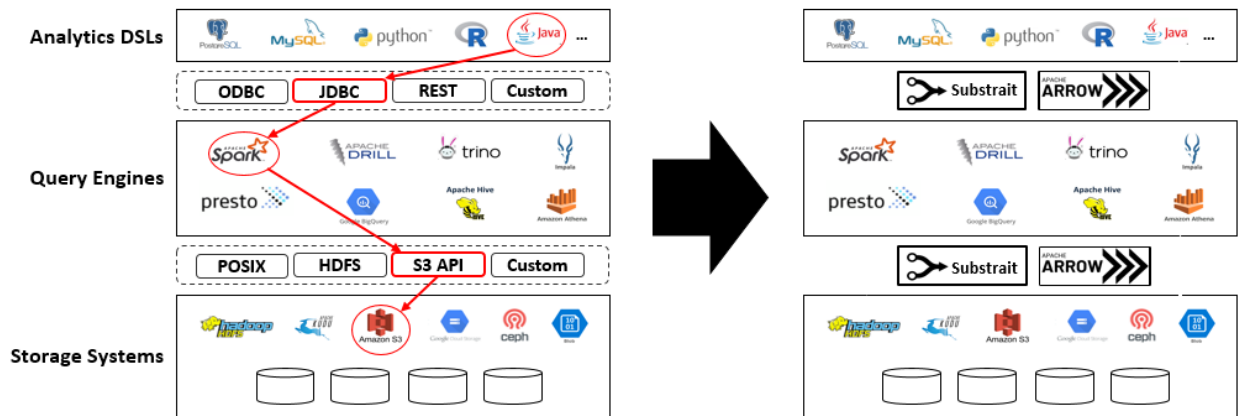


Figure 35: Flexibility Offered by Substrait and Apache Arrow

⁶⁰ Cloud Object Storage – Amazon S3: <https://aws.amazon.com/s3/>

⁶¹ S3 Select Pushdown: <https://docs.aws.amazon.com/emr/latest/ReleaseGuide/emr-presto-s3select.html>

8.3.2.4 Object-based Computational Storage System

The Object-based Computation Storage⁶² (OCS) System, as shown in Figure 36, leverages both Object Interface and Substrait to offer the same data view and offload computation operations respectively. In the OCS storage stack, an analytics client prepares and transmits the Substrait query plan to the OCS frontend server. The frontend server identifies computational operations supported by the backend Object-based Computational Storage Array⁶³ (OCSA) and splits the query plan into two parts:

- Pushdown plan that contains operations performed by OCSA; and
- Remainder plan executed by the client.

OCS leverages the Object Interface to transmit the Pushdown query plan to OCSA. OCSA processes the Pushdown plan and returns the query result back to the frontend, which is returned back to the client to execute the remainder plan. Overall, the OCS system significantly reduces data movement across the network and improves query execution time.

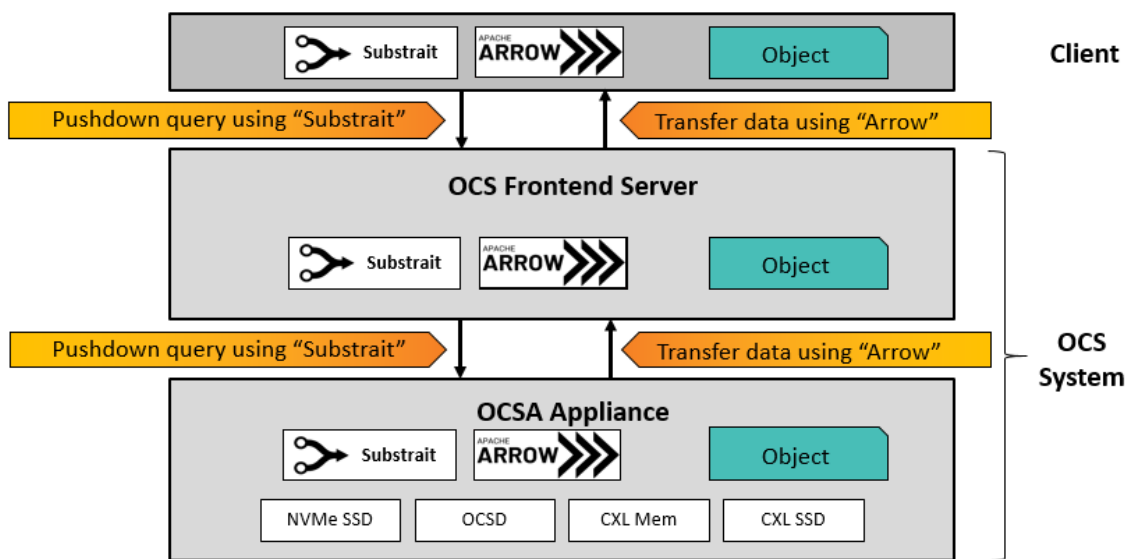


Figure 36: Object-based Computation Storage (OCS) System

8.3.2.5 H.264 encode use case

The "H.264 encode" use case is a practical application of the SNIA Computation Storage API. This use case is derived from "A.1, Initialization and queuing a synchronous request". The API requires several arguments, including the CSx name for H.264 encoder, host memory for source/destination, JSON file containing the parameters for this compute function, and status monitoring. This use case uses SNIA CS API functions including csOpenCSx, csAllocMem, and csQueueComputeRequest. The use case offloads H.264 encoding to a device, thereby reducing the load on the host system.

To encode video data, the "H.264 encode" function accepts several parameters, including a path to a file or directory that resides on a device, the name of the device, host memory for input and output data, the name of a JSON file containing all the parameters for the H.264 encoder, and a status to

⁶² Accelerating Data Analytics Using Object Based Computational Storage in HPC:
https://sc23.supercomputing.org/proceedings/exhibitor_forum/exhibitor_forum_pages/exforum116.html

⁶³ Accelerating Data Analytics Using Object Based Computational Storage in HPC:
https://sc23.supercomputing.org/proceedings/exhibitor_forum/exhibitor_forum_pages/exforum116.html

keep track of all the error conditions. The function can be used for various applications, ranging from low-delay, low-bitrate mobile transmissions to HD or FullHD consumer TV, to even professional television productions. To make the function more versatile, other use cases like asynchronous command request and batch processing can be made into a single API in a similar manner.

9 Management

Redfish provides a RESTful management interface for managing compute elements. While this has not been specifically developed for PIM or PNM, this interface may be expanded for this model.

9.1 Redfish

Redfish⁶⁴ is an industry standard software defined management technology that provides a common data model and APIs for discovering, monitoring, and managing these resources . It is designed to deliver simple and secure management for converged, hybrid IT and the software defined data center (SDDC). The standard defines a protocol that uses RESTful interfaces to provide access to data and operations associated with the management of systems and networks. One of the strengths of the Redfish protocol is that it works with a wide range of servers, from stand-alone servers to rack-mount and bladed environments to large-scale data centers and cloud environments. The Redfish standard addresses several key issues for infrastructures that require scalability. Large infrastructures often consist of many simple servers of different makes and types. This hyper-scale usage model requires a new approach to systems management. The Redfish protocol addresses these needs by providing a standard protocol based on out-of-band systems management.

A Redfish service, or simply service, is a software or firmware product that implements the protocols, resources, and functions of the Redfish Specification. In many cases, a baseboard management controller (BMC) implements Redfish to provide remote management capabilities of a system. A service implements resources in the Redfish data model at different Universal Resource Identifiers (URIs). All services support an entry point called the service root located at the URI /redfish/v1/. The service root contains references to the supported top-level resource collections and services supported by the implementation. Hyperlinks in responses from the service are used to guide clients to other resources in the service, which contain the URI of the referenced resource. Figure 37 shows a simplified resource map for managing a server.

⁶⁴ Redfish documentation, Distributed Management Task Force www.dmtf.org

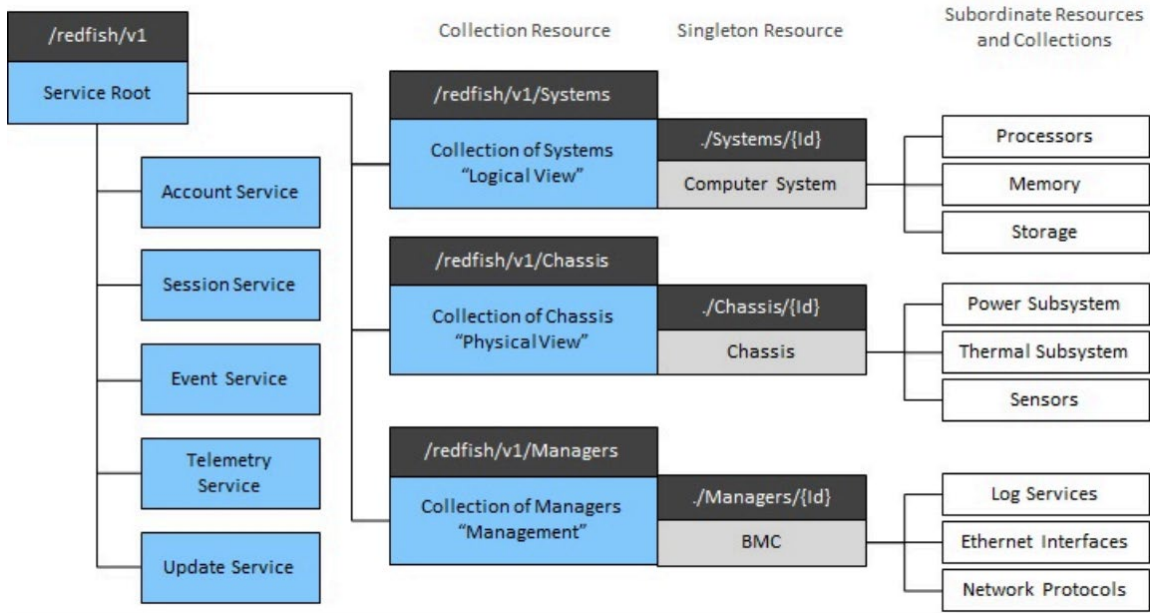


Figure 37: Redfish Resource Map

There are three commonly used sets of resources for managing a server, namely, Chassis, Managers and Systems. Within each resource collection are instances that represent specific devices or components. E.g., the Systems collection contains instances of each server in the data center. Each resource instance contains metadata and properties and may include serial number, manufacturer, CPU details, memory configuration, power status etc. A resource may also have actions that may be performed on the resource instance such as power On/Off a server, reset a management controller, or create a virtual drive on a storage array. Resources may also cross reference each other such as a server being associated with a chassis.

9.1.1 Discovering with Redfish

Individual resources may be discovered using Redfish by querying and walking the URI schema. The following example describes how a GPU resource may be discovered as shown in Figure 38. Starting at the Redfish Service Root **/redfish/v1**, the Redfish API through the RESTful interface is used to query the Systems resource collection, and identify the specific system of interest, e.g., **Server-x1**. Next, query the PCIe devices under the PCIeDevices collection using URI **/redfish/v1/Systems/Server-x1/PCleDevices**. The PCIeDevices collection provides a list of all PCIe devices such as storage controllers, network cards, GPUs etc. Each GPU found may be further queried for detailed information on corresponding properties that include a name, manufacturer, model, serial number, part number, firmware version, memory, status, etc. A specific GPU is then selected from the list from one or more GPUs found.

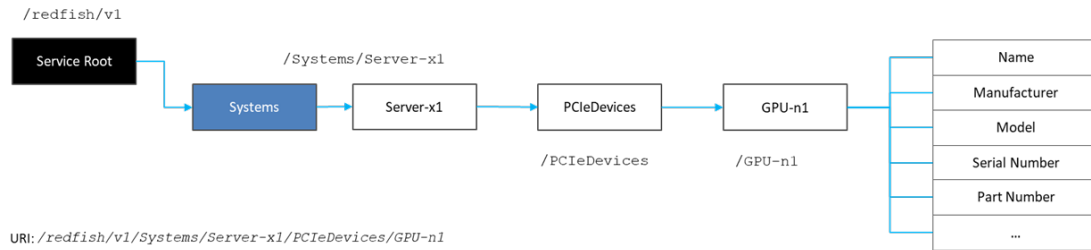


Figure 38: Discovery example using Redfish

The following command is an example of querying for a GPU device, where, SystemId represents the server Server-X1, and DeviceId represents the GPU:

```
curl -k https://<redfish_host>/redfish/v1/Systems/{SystemId}/PCIEDevices/{DeviceId}
```

9.1.2 Challenges with Redfish

Redfish has an exhaustive list of use cases for efficient management such as BMC and BIOS configuration, inventory queries, service provisioning, scaling and load balancing, configuration management, storage management, health monitoring, event logging, power management, maintenance, security, and user accounts. There are still challenges in how a device may be discovered. Lets take the example of a GPU above, which may also be discovered under the URI /redfish/v1/Systems/Server-x1/Processors due to the cross-referencing ability of the schema, where, it identifies itself with other processor types and as a GPU ProcessorType. Similarly, a DPU may be discovered under Processors, as well as under NetworkAdapters when it is a SmartNIC. However, an In-Memory Processing device is located under the Memory collection with additional attributes that indicate it to be compute capable. Each of these examples show that NDC devices may be spread out in different locations that a user has to be aware of. This may limit the discovery of new derivatives of NDC devices. There is no one solution to solve the discovery and management of NDC devices. While Computational Storage devices may be discovered using NVMe and SNIA CS API standards, it does not expand to other near memory compute device types. OCP has been working on standardizing GPU and accelerator management interfaces⁶⁵ and RAS requirements⁶⁶ to address limitations in them across different vendors. But this does not address NDC devices. Redfish may be enhanced to provide a schema that allows efficient discovery and management of NDC devices, and may be in the best position to be most effective.

⁶⁵ OCP GPU & ACCELERATOR MANAGEMENT INTERFACES <https://www.opencompute.org/documents/ocp-gpu-accelerator-management-interfaces-v0-9-pdf>

⁶⁶ OCP GPU & ACCELERATOR RAS REQUIREMENTS V0.9 <https://www.opencompute.org/documents/ocp-gpu-and-accelerators-ras-requirements-v0-9-pdf>

10 Conclusion

With the massive increase of data, it is critical that we observe data gravity and move compute near data to reduce power and increase performance. There are a number of types of NDC that facilitate moving compute near data and the appropriate NDC technology should be selected for the application and the location of the data.

One embodiment of NDC is Computational Storage (CS) which has been around for a number of years. Some of the lessons learned from CS can be extended to all forms NDC deployment. In CS, there are numerous application specific implementations that are tailored for their specific application. These implementations are not usable by a generic user that wishes to perform their own specific compute at the device that is different from the narrowly focused compute provided in the design.

One of the barriers to widespread adoption of NDC is the lack of a standardization for the various NDC technologies we touched on. Common standardized mechanisms are required to discover, configure, manage, program and utilize NDC. There is work being done to improve the programming environment for NDC. As this programming environment is developed, NDC will be more accessible for a breadth of applications. Future development of a more generalized form of programming NDC devices will facilitate broader market adoption.

11 Acronyms

AI	Artificial Intelligence
API	Application Programming Interface
CRC	Cyclic Redundancy Check
CSA	Computational Storage Array (see 2.1)
CSD	Computational Storage Drive (see 2.1)
CSF	Computational Storage Function (see 2.1)
CSP	Computational Storage Processor (see 2.1)
CSx	Computational Storage Device (see 2.1)
CXL	Compute Express Link
DDR	Double Data Rate
DL	Deep Learning
DMA	Direct Memory Access
DPU	Data Processing Unit (see 2.4)
DRAM	Dynamic Random Access Memory
FPGA	Field Programmable Gate Array
GDDR	Graphics DDR
GPU	General Processing Unit
HBM	High Bandwidth Memory
HPC	High Performance Computing
KV	Key Value
LBA	Logical Block Address
LPDDR	Low Power DDR
LSM	Log Structure Merge
LVM	Logical Volume Manager
MCC	Memory Coupled Computing
ML	Machine Learning
NAND	Shorthand for the common form of Solid Storage Medium
NDC	Near Data Computing (see section 2)
NIC	Network Interface Card
NPU	Neural Processing Unit
P2P	Peer-to-Peer
PE	Processing Element
PE	Processing Element
PIM	Processing-In-Memory (see 2.2)
PNM	Processing-near-memory (see 2.3)
RDMA	Remote DMA
ROCm	Radeon Open Compute platform (see 4.3.1.2)
Smart Switch	Network Switch that contains processing capabilities
SmartNIC	Network Interface Card that contains processing capabilities
SoC	System-On-Chip
SSD	Solid State Drive
TCO	Total Cost of Ownership
TPU	Tensor Processing Unit
URI	Universal Resource Identifier
WAF	Write Amplification Factor
ZNS	NVMe Zoned Namespace Command Set

12 License

Creative Commons

OCP encourages participants to share their proposals, specifications and designs with the community. This is to promote openness and encourage continuous and open feedback. It is important to remember that by providing feedback for any such documents, whether in written or verbal form, that the contributor or the contributor's organization grants OCP and its members irrevocable right to use this feedback for any purpose without any further obligation.

It is acknowledged that any such documentation and any ancillary materials that are provided to OCP in connection with this document, including without limitation any white papers, articles, photographs, studies, diagrams, contact information (together, "Materials") are made available under the Creative Commons Attribution-ShareAlike 4.0 International License found here: <https://creativecommons.org/licenses/by-sa/4.0/>, or any later version, and without limiting the foregoing, OCP may make the Materials available under such terms.

As a contributor to this document, all members represent that they have the authority to grant the rights and licenses herein. They further represent and warrant that the Materials do not and will not violate the copyrights or misappropriate the trade secret rights of any third party, including without limitation rights in intellectual property. The contributor(s) also represent that, to the extent the Materials include materials protected by copyright or trade secret rights that are owned or created by any third-party, they have obtained permission for its use consistent with the foregoing. They will provide OCP evidence of such permission upon OCP's request. This document and any "Materials" are published on the respective project's wiki page and are open to the public in accordance with OCP's Bylaws and IP Policy. This can be found at <http://www.opencompute.org/participate/legal-documents/> . If you have any questions, please contact OCP.

Date: August 19, 2025



This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

13 About Open Compute Project

The Open Compute Project (OCP) brings at-scale innovations and hyperscaler best practices to all, spanning technology domains from the data center to the edge, and the technology stack from silicon, to systems, to site facilities and services. The international OCP Community is made up of organizations and people from hyperscale and tier-2 cloud data center operators, communications providers, colocation providers, diverse enterprises, and technology vendors. The OCP Foundation fosters collaboration within the Community to tackle market challenges in an array of OCP Projects that create open contributions such as specifications, designs, and more. The OCP Solution Provider Program then recognizes the application of those contributions in compliant products, solutions, and data center facilities and services with designations like OCP Inspired, OCP Accepted and OCP Ready. With the tenets of openness, impact, efficiency, scale and sustainability, the OCP engages and educates thousands of engineers every year through many events and webinars. Across many initiatives the OCP Foundation and Community are meeting the market today and shaping the future.

Learn more at: www.opencompute.org.

