Datenbank-Spektrum
https://doi.org/10.1007/s13222-026-00564-6

FACHBEITRAG

Near-Data Processing in CHORYS

Philippe Bonnet[1] · Alberto Lerner[2] · Davide Zoni[5] · Dirk Habich[3] · Erwann Kervennic[6] · Gianni Antichi[5] · Giovanni Grandi[8] · Johannes Pietrzyk[3] · Martijn Wokke[7] · Noa Omer[7] · Nuno Neves[9] · Nuno Roma[9] · Pedro Tomas[9] · Wolfgang Lehner[3] · Zsolt István[4]

Received: 1 June 2026 / Accepted: 20 September 2026
© The Author(s) 2026

Abstract
Data-intensive applications in the cloud are expected to service an ever-increasing number of users and handle exponentially growing volumes of data. Meeting these expectations is a challenge because they require balanced performance across the full memory hierarchy, from processor caches to disks accessed over the network. The problem is compounded by the cloud's shift toward disaggregated architectures, where memory and storage are pooled across fabrics, thus lengthening the data path. CHORYS addresses this challenge through near-data processing with open, programmable accelerators embedded in storage and network devices. CHORYS considers an accelerator-based cloud model in which Solid State Drives (SSDs) and Network Interface Cards (NICs) execute data-intensive operations such as filtering, aggregation, and parsing directly on the data path, thus reducing latency, energy consumption, and resource usage. The project contributes both hardware platforms (OpenSSD-V and OpenNIC-V, built around RISC-V cores) and a software stack including data-intensive pattern libraries, APIs, and runtime systems for the seamless integration of near-data execution into existing cloud infrastructures. CHORYS use cases illustrate near-data processing in the context of data lakes, object stores, and AI inference pipelines.

Keywords Near-data processing · Solid-state drives · SmartNIC · Hardware accelerator · Cloud

1 Introduction

Data-intensive applications in the cloud are expected to service an ever-increasing number of users and handle exponentially growing volumes of data. This raises several system problems linked to the traditional issue of incommensurate scaling [1]. First, compute and storage scale at different paces. Second, the multiple layers of the storage hierarchy do not scale together in terms of bandwidth or latency. As a result, data-intensive applications in the cloud are increasingly bottlenecked not by compute but by the cost of moving data. Energy, latency, and bandwidth budgets are dominated by the traffic between processors and the memory and storage tiers that feed them.

Disaggregation is a solution to the first problem. Disaggregated memory and storage are moved off the server into pooled tiers accessible over a fabric such as CXL or RDMA. This way, capacity and compute can scale independently. In addition, resources do not remain unused on servers where they are not needed. However, disaggregation makes the data movement problem worse, not better. A simple LOAD instruction that used to take around tens of nanoseconds to local DRAM now takes a few hundred nanoseconds over CXL or microseconds over a network fabric. Elasticity and utilization are gained, but at the cost of a longer data path.

Near-data processing is a way to reduce that cost. Near-data processing leverages processing elements located in or next to memory and storage—processing-in-memory, computational SSDs, smart NICs, accelerators on memory expanders—so that operations execute close to the data they touch, thus reducing data movement to a host

✉ Philippe Bonnet
bonnet@di.ku.dk

1 University of Copenhagen, Copenhagen, Denmark
2 Computing Flows, Fribourg, Switzerland
3 TU Dresden, Dresden, Germany
4 TU Darmstadt, Darmstadt, Germany
5 Politecnico di Milano, Milano, Italy
6 Menta, Valbonne, France
7 Cyso, Alkmaar, The Netherlands
8 Codasip, Munich, Germany
9 INESC-ID, Lissabon, Portugal

Published online: 26 October 2026

CPU. Even modest compute along the data path—a small RISC-V core, an FPGA, fixed-function accelerators, PIM-style logic—facilitates operations that would otherwise require shipping large amounts of data back to the host to be executed, to reduce data movement. Common targets include scans, filters and aggregations in databases, embedding-table lookups and reductions in recommendation models, pointer chasing in graphs, bulk memory operations, and compression and encryption.

Current trends (including CXL memory pools, computational memory, near-memory accelerators on memory expanders, or smart memory nodes) show the two ideas converging. Disaggregation provides the substrate, a shared, fabric-attached memory and storage tier, and near-data processing provides the mechanism that makes that substrate performant, by keeping work close to bytes rather than shuttling bytes to work. Disaggregation, in other words, creates the conditions under which near-data processing stops being a niche optimization and becomes structurally necessary.

In this paper, we give an overview of the vision of near-data processing developed by the CHORYS project. The goal of this project is to develop open, programmable accelerators built around RISC-V cores embedded in storage and network devices that execute data-intensive operations directly on the data path. We present near-data processing in the context of the cloud architecture, briefly describe the characteristics of the artifacts to be developed in the project (OpenNIC-V, OpenSSD-V, and GranForge), and discuss three potential use cases.

CHORYS is a Horizon Europe research and innovation action, funded under the call HORIZON-CL4-2024-DIGITAL-EMERGING-01-21, which supports prototypes of cloud and edge servers built on European processor technology and aims to consolidate a reference open cloud infrastructure—including standards, interfaces, and best practices. Together with its two sibling projects, HIGHER[1] and CAPE[2], CHORYS develops modular, hardware-software co-designed environments rooted in European hardware. Within this cohort, CHORYS contributes the near-data processing angle. Project information is published on the project website at chorys.eu, and deliverables, publications, datasets and open-source artifacts are released through the project's Zenodo community at zenodo.org/communities/chorys.

2 Related Work

The case for moving computation closer to data has grown more compelling as the field confronts the slowdown of Moore's law and the end of Dennard scaling. Barroso et al. [2]. contend that heterogeneous computing (combining general-purpose processors with a diverse set of accelerators) is the most promising response to this challenge at warehouse scale. Dally et al. articulate the principles that govern the design of domain-specific hardware accelerators [3]. CHORYS focuses this broad agenda on *data-intensive* applications, for which the bottleneck is access to data, rather than the supply of compute cycles [4, 5].

Accelerators in Production Clouds Major US and Chinese cloud operators introduced accelerators into their infrastructure years ago. Microsoft's Catapult architecture placed FPGAs between network switches and servers to accelerate both infrastructure functions and search ranking [6]. The company subsequently relied on FPGAs for deep-learning inference [7] and on smart NICs to offload the networking stack [8]. Google deployed smart NICs to accelerate remote memory access [9], while AWS offloads hypervisor and I/O management to its Nitro cards and specializes them in Redshift to perform near-data processing, reportedly improving performance by up to an order of magnitude [10]. Alibaba pairs SSDs with embedded compression engines to cut storage costs in PolarDB [11] and employs smart NICs for congestion control [12].

These deployments confirm the shift toward the accelerator-based data center anticipated by VMware Project Monterey [13] and Intel [14]; however, they remain fully proprietary, so their benefits accrue only to the providers themselves. This gap motivates the CHORYS focus on *open* and programmable accelerators usable by cloud providers and customers alike.

RISC-V-based Accelerators A parallel line of work pursues open instruction sets for accelerators. Commercial RISC-V accelerators already span networking and AI inference, and the RISC-V community has identified the need for a dedicated accelerator working group to define a common ecosystem [15]. For instance, the nanoPU prototype co-designs a NIC and a CPU to slash the latency of RPC-heavy cloud workloads [16]. Where most existing efforts target compute-bound or networking tasks, CHORYS instead aims at reducing data movement for recurring data-intensive patterns on RISC-V cores.

ISA Extensions for Data Movement The canonical RISC-V extension is the vector extension, realized either as a loosely coupled external unit, as in the open-source New Ara design [17], or as a tightly integrated unit, as in the Andes AX45MPV core [18]. For the specific problem of data movement, Intel's Accelerator Interfacing Architecture exposes instructions that offload bulk transfers and synchronizes with accelerators through dedicated work queues [19],

[1] https://www.higher-project.eu.

[2] https://cape-project.eu.

backed by a dedicated data-mover library [20]. Comparable, movement-oriented ISA extensions are not yet available for RISC-V, which is the gap CHORYS sets out to close with functional units and instructions tailored to near-data processing.

Open and Programmable SSDs and NICs Programmable NICs are already widely deployed, and the segment is projected to surpass $ 5 billion by 2028 [21], whereas programmable SSDs are still in an early stage of maturity [22]. A data-centre NVMe SSD, such as the Samsung PM9A3 [23, 24], hides several embedded ARM Cortex-R8 cores [25] behind a fixed-function controller, dedicating substantial energy to persistence alone. Cloud providers have long designed their own storage to match workload requirements [26], and computational storage—attaching a programmable compute unit to a drive—has since been standardized by SNIA and NVMe [27]. Crucially, how an accelerator is integrated with the rest of the system often matters more than the accelerator itself [28]. Existing open platforms, however, were built for prototyping rather than code offload: Cosmos+ OpenSSD [29] on the storage side and NetFPGA [30] on the networking side. Commercial RISC-V storage controllers such as Alibaba's Zhenyue 510 [31] and Phison's X1 [32] are ASICs whose functionality is fixed. CHORYS instead targets open, RISC-V-based, programmable, and computational devices, building on revamped OpenSSD components for storage [33, 34], and on open NIC substrates—Xilinx OpenNIC [35], Corundum [36], and NetFPGA [30]—for the network path.

Programming Models for Accelerators Finally, accelerators must be programmable in a hardware-agnostic way. There are two dimensions to this issue:

1. How to program accelerators?
2. How to incorporate accelerators into a system?

In terms of programming language for accelerators, P4 has been widely adopted on Smart NICs, with its simple match-action rules model well-suited for packet handling [37]. On computational storage, eBPF was proposed as a vendor-neutral representation of downloadable programs [27], but this solution has not been adopted by industry. On NVIDIA Data Processing Units, DOCA is the programming framework that must be used [38]. However, it has well-documented limitations [39].

In terms of the integration of accelerators into a system, generic frameworks such as SYCL [40] and oneAPI [41] have been proposed with a unified memory and asynchronous-execution model spanning host and devices. Such frameworks ease portability but compare poorly to vendor-specific solutions in performance [42] and are arguably too generic for near-data processing.

The Template SIMD Library pioneered at TU Dresden provides a means to combine both aspects via specialization. TSL is a tool that generates hardware-agnostic C++ function templates specialized for a given SIMD target [43]. CHORYS generalizes this template-library approach to data-intensive patterns for near-data processing on SmartNIC and computational storage.

3 Architecture

CHORYS assumes a concrete cloud architecture into which open, programmable storage and network accelerators can be introduced without disrupting existing infrastructure. We describe this architecture top-down, across three levels of detail. We first present the physical deployment model of an accelerator-based rack server (Section 3.1). We then describe the logical software stack through which applications and operators access the accelerators (Sections 3.2 and Sect. 3.3) as well as the data flow that this stack enables inside a server (Section 3.4). Finally, we zoom into the devices themselves and characterize the design space for coupling their RISC-V cores with data-pattern accelerators (Section 3.5). The guiding principle throughout is drop-in compatibility: a cloud provider should be able to map every CHORYS component onto a component it already operates, and allow near-data processing incrementally rather than through disruptive architectural change [44].

3.1 Deployment Model

The deployment target is a standard rack-mounted cloud server, the building block of warehouse-scale computers [2], augmented with the project's two I/O devices: OpenNIC-V (Section 4.2) in place of, or alongside, a conventional network interface card, and OpenSSD-V (Section 4.1) in place of, or alongside, a conventional NVMe solid-state drive. No specialized chassis, interconnect, or non-standard host component is assumed; the model deliberately follows the Open Compute Project rack and tray specifications[3], so that accelerated and conventional devices share the same PCIe interconnect and the same I/O and memory hierarchy.

Each server retains its conventional host components—multi-core CPUs, main memory, a PCIe fabric, and optional GPUs for inference and training—while OpenNIC-V sits on the network ingress and egress path and OpenSSD-V on the persistent-storage path. The host CPU remains responsible for control-plane logic, orchestration, and application coordination, whereas the two devices execute selected data-plane operations on their embedded RISC-V cores and spe-

[3] https://www.opencompute.org/projects/server.

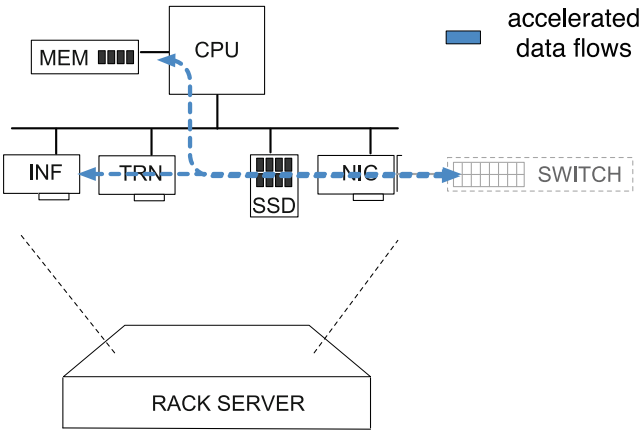

Fig. 1 Rack server with CPU, memory (MEM), accelerators for inference (INF) or training (TRN), highlighting the accelerated data flows that traverse OpenSSD-V (SSD)/OpenNIC-V (NIC)

cialized accelerators. This placement yields three properties on which the project depends: the accelerators occupy the critical data paths for storage and network traffic; host-side execution and GPU workloads are unchanged at the hardware level; and multiple accelerators can coexist in one server to form end-to-end accelerated pipelines. At rack scale, the model composes naturally under a conventional top-of-rack switch, optionally grouping OpenSSD-V drives in a storage chassis[4] and housing OpenNIC-V in hot-swappable NIC modules (Fig. 1).

3.2 Logical Cloud Stack

A traditional cloud application is organized as a layered stack: user-visible *applications* (databases, message queues) rest on purchasable *external services* (object and block stores, group communication), which in turn rest on proprietary *internal services*—distributed file systems [45], lock servers [46], and logging and remote-procedure-call machinery—and finally on a commoditized hardware/OS base layer running Linux or a hypervisor [2]. CHORYS introduces its devices at this base layer and lets them export services upward, so that an accelerated stack runs in parallel with, and improves, the conventional one.

To this end, the logical software stack spans from the OpenNIC-V and OpenSSD-V hardware up to the demonstrator software, with two goals: to hide device-specific detail behind stable interfaces, and to emulate the essential structure of a production cloud stack without rebuilding it. At the lowest level, the devices run programs on their RISC-V cores and accelerators, operating directly on storage and network streams. Above them, a device-side runtime and driver layer—Menshen [47] for the network card and an SSD-resident operating system for the drive—handles program loading, I/O coordination, and memory management, exposing the accelerators as programmable cloud resources rather than fixed-function devices. On top of the runtime, *programming APIs* let developers target the cores and accelerators with dataflow abstractions (filtering, parsing, aggregation, data-structure transformation), while *management APIs* expose device configuration, monitoring, and isolation to the host control plane.

The benefit propagates upward by composition. A distributed file system whose read/write API is extended to run a program on the device becomes near-data-processing-enabled; an object store layered on that file system can then expose computation to its clients—an idea previously attempted, then withdrawn, by Amazon S3 Select; and a data-lakehouse service [48] layered on the computational object store can push SQL operators down to the device, the scenario the project refers to as SQL-over-Parquet, after the columnar file format it targets[5]. The same composition supports machine-learning pipelines, in which image pre-processing that is normally performed on the host CPU [49] can instead be expressed as device-side data patterns (Fig. 2).

3.3 Management Stack

Cloud operators run a management stack that mirrors the application stack. Its user-visible layers—billing and dashboards—rest on a scheduler that orchestrates the tens to hundreds of thousands of processes that make up a service [50, 51] and on a monitoring-and-alerting service; these in turn rest on binary-distribution and telemetry services [52], and finally on a base layer providing resource management, binary loading, and process introspection.

CHORYS extends this base layer to treat its devices as managed computing elements, without altering the layer's function: resource limits cap how much of a device a program may use, a binary loader launches device programs, and process introspection reports their operational data. The

[4] https://www.opencompute.org/projects/storage.

[5] https://parquet.apache.org.

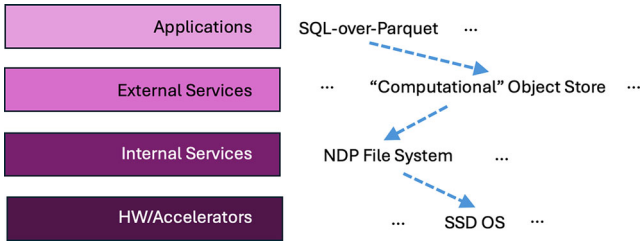

Fig. 2 The layered cloud stack and how accelerator features exposed at one layer support features at the next

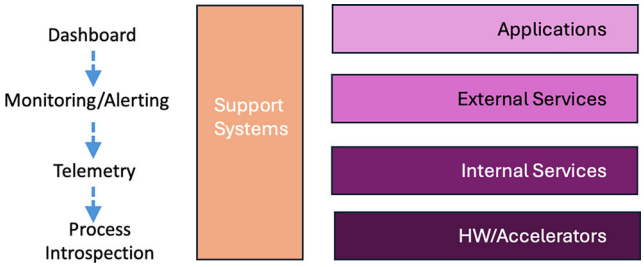

Fig. 3 The management stack and its extension so that resource limits, binary loading and process introspection apply to accelerators

difference from a CPU is that a device program attaches to one or more naturally occurring data flows rather than controlling its own data access. Consequently, resource consumption is expressed as flow bandwidth, binary loading must name the flows a program attaches to, and introspection reports the rate at which a flow traverses the device and whether the program keeps up. These primitives compose upward exactly as in the application stack: telemetry probes a device for its per-program processing rate, a monitoring/alerting service decides whether that rate is adequate, a dashboard surfaces it [53], and internal counters can feed a provider's billing model. Programming the devices thus differs from programming CPU-based servers, but managing them does not (Fig. 3).

3.4 Server- and Device-Level Data Flows

The architectural payoff is visible in the data flows. In a conventional server, every flow is mediated by the host CPU: data read from storage or received from the network is staged in host memory before being forwarded to a GPU or written back, so the CPU becomes a bottleneck, and the energy spent moving data can exceed that of the computation itself—a cost repeatedly identified as a limiting factor for efficient systems [3, 54]. Because OpenSSD-V and OpenNIC-V are designed as drop-in replacements that nonetheless compute, a transformation that previously required two transfers (from I/O to the CPU and from the CPU to a GPU) may now be reduced to a single transfer from the device to its consumer. For *reductive* operations such as filtering and aggregation, whose output is smaller than their input, shifting the computation to the device also shrinks the volume that must be moved at all [55].

At the device level, near-data processing is realized by logically attaching a program to a flow. A read on OpenSSD-V, for example, can be intercepted and routed through a pipeline that parses the Parquet stream with a data-pattern accelerator, filters the reconstructed records on the RISC-V core, and re-encodes the surviving records with a deparser accelerator, so that the read returns the result of an SQL pushdown rather than the raw bytes. Combining a RISC-V core with such accelerators is deliberate: a core fast enough to rival server CPUs would dominate the device's energy budget, whereas specialized accelerators are orders of magnitude more efficient [3, 56] but too narrow on their own. Pairing a modest, workload-tuned core with data-pattern accelerators captures the efficiency of the latter while retaining the generality of the former [5, 28].

3.5 Coupling RISC-V Cores and Accelerators

The efficiency just described hinges on how tightly the RISC-V core and the accelerators are coupled inside each device—a hardware-design concern realized in OpenSSD-V and OpenNIC-V (Sections 4.1 and Sect. 4.2) and driven by the data-intensive patterns that GrandForge defines (Section 4.3).

The design space is organized along two dimensions. The first is the *placement* of an accelerator relative to the core: an *integrated* accelerator is disguised as a CPU functional unit, whereas a *stand-alone* accelerator is an external unit that exchanges data with the core. The second is the *plane* on which the core interacts with the accelerator: the data path, where the core feeds or drains operands, or the control path, where the core configures and supervises the unit. The two dimensions yield four quadrants, each calling for a different class of RISC-V extension. The primitives that motivate these extensions are common data-manipulation operations such as sorting, naturally realized by a bitonic network [57], and the transposition required when database records switch between row- and column-oriented layouts [58].

Integrated accelerators on the data path are exposed through new or redefined instructions, typically surfaced to the compiler as intrinsics. Some reuse existing registers: the RISC-V vector extension is the canonical case, whether realized as a loosely coupled unit [17], a tightly integrated one [18], a micro-programmable variant [59], or a width-scaled design [60]; the same approach accelerates pointer chasing over linked structures by reusing the scalar registers [61]. Others must add registers: a matrix coprocessor placed close to data [62] relies on systolic-array techniques that stream operands without spilling intermediates back to the register file at every instruction [63].

Stand-alone accelerators on the data path are reached not through instructions but through special registers that mediate a stream between core and unit, with side effects on read and write. The nanoPU exposes received and transmitted network data through such registers [16], and stream semantic registers turn sequences of loads and stores into operations on a streaming unit that is never exposed to the programmer [64]. On the control path, integrated accelerators benefit from low-latency signalling such as userspace in-

terrupts and timers, which let an application overlap useful work with a long-running accelerator without trapping into the kernel [65]; stand-alone accelerators instead need efficient parameter mapping, for which a hardware sequencer can replace the storm of per-task register writes that would otherwise saturate a high-IOPS device. Folding such extensions into a core is itself an open choice among integration interfaces such as RoCC [66], SCAIE-V [67], and CV-X-IF [68]; CHORYS defers the selection of concrete extensions and their integration to later design iterations, once the use cases and device prototypes have matured.

4 CHORYS Artifacts

The architecture of Section 3 is realised by three artifacts developed in the project: two open, programmable I/O devices, OpenNIC-V on the network path and OpenSSD-V on the storage path, and GrandForge, a library of data-intensive patterns that spans the host and, ultimately, the devices. The two devices share a common substrate (illustrated in Fig. 4, see for details): RISC-V softcores integrated with data-pattern accelerators on an FPGA, specialised per workload through device-specific and downloadable functions [69]. GrandForge supplies the software counterpart, defining the operators that the devices accelerate and the stable plan interface through which the rest of the cloud stack reaches them [70]. We outline each in turn [71].

4.1 OpenSSD-V

OpenSSD-V is an open, programmable computational SSD that computes on data as it is read or written rather than merely serving blocks in a predefined way. Conventional SSDs are black boxes whose undocumented, proprietary hardware and firmware components. We revisit the Cosmos+ OpenSSD platform [29] and the broader line of work on revamped OpenSSD components [33, 34], replacing the fixed-function controller with RISC-V softcores and data-pattern accelerators. The same RISC-V softcores considered for OpenNIC-V apply here, paired with an SSD-resident runtime that manages program loading, I/O, and the flash translation layer; cache-coherence protocols across cores or over CXL are explicitly out of scope [69]. The device's defining capability is to guide NVMe commands (or possibly new forms of host-device interactions) through an accelerated pipeline, e.g., parsing a Parquet stream, filtering the reconstructed records on the core, and re-encoding the survivors so that the I/O returns the result of a pushed-down operation rather than raw bytes in a *Parquet SSD* (see Section 5) [72].

4.2 OpenNIC-V

OpenNIC-V is an open, programmable smart NIC that turns the network interface into a near-data accelerator. It builds on the Xilinx OpenNIC shell [35], a documented baseline whose data and control planes are extended with RISC-V softcores and the data-pattern accelerators outlined in Section 3.5. The reference platform is the AMD U55C board, which carries a Virtex UltraScale+ FPGA with 16 GB of HBM2, a PCIe Gen4 host link, and two 100 Gb QSFP28 ports; its ample logic and memory make it well suited to NIC exploration, although it provides no direct connectivity to flash modules [69]. The candidate softcores are the Codasip A730—a 64-bit in-order dual-issue core with a floating-point unit, whose vendor toolchain permits custom ISA extensions—and the open-source Snitch cluster [73], a configurable group of lightweight 32-bit RISC-V cores connected through an AXI crossbar; the suitability

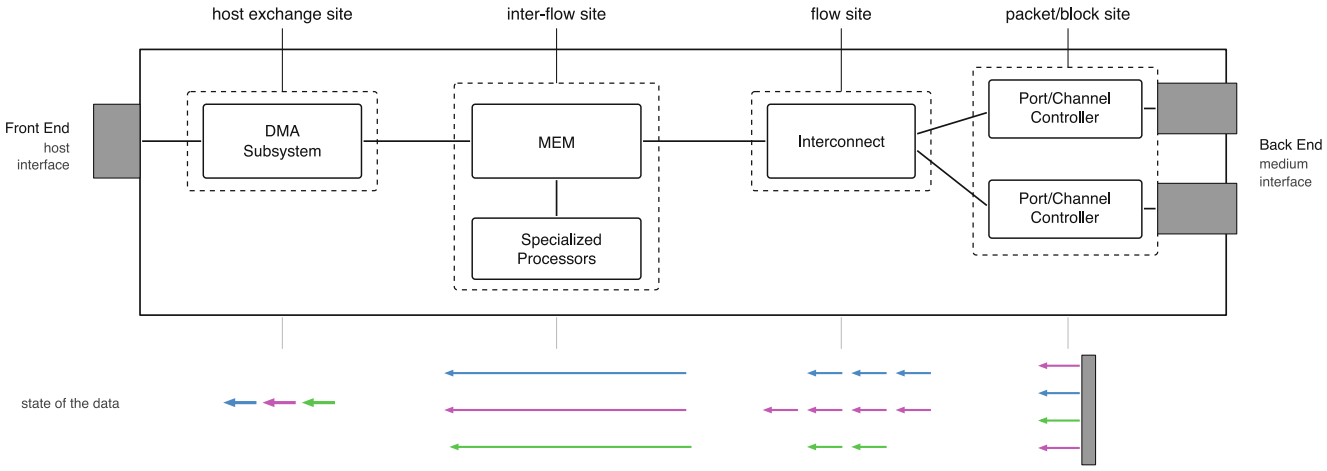

Fig. 4 Generic platform design for OpenSSD-V and OpenNIC-V, with four compute sites placed along the path between the host interface and the medium interface (above), together with a logical view of the data levels inside a storage or network device (below)

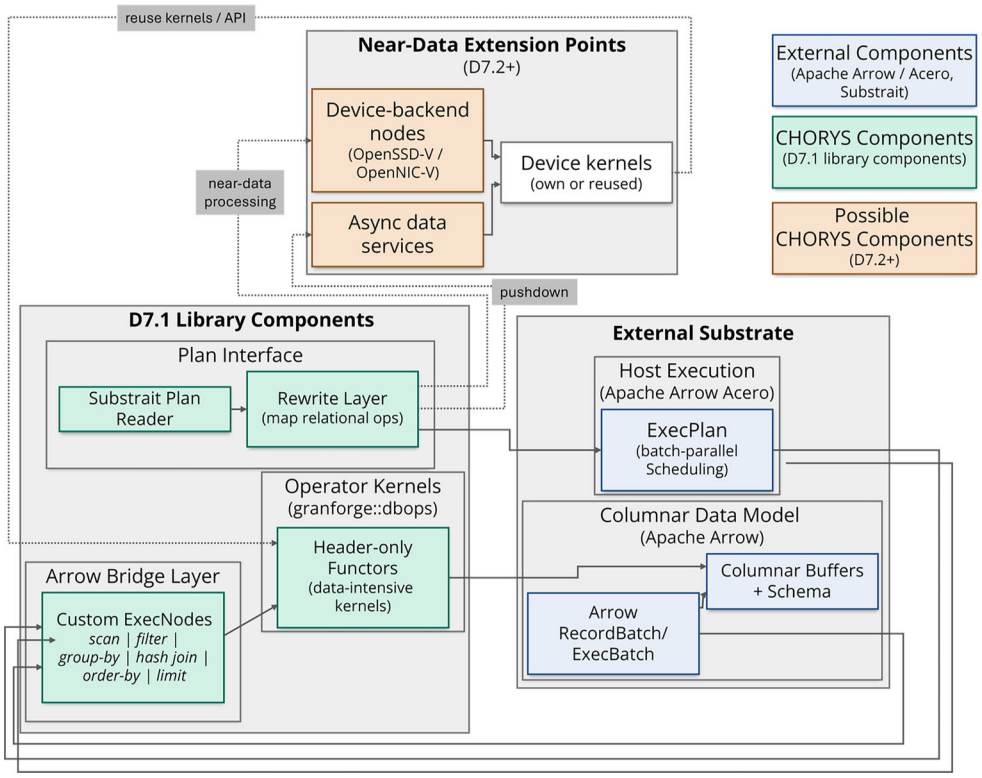

Fig. 5 Granforge library architecture and external components

of either for the tight latency budgets of a smart NIC remains an open question [69]. A device-side runtime based on Menshen [47] loads programs, coordinates I/O, and exposes the accelerators as managed cloud resources (Section 3.3). On this platform OpenNIC-V executes reductive sub-operators (e.g., filtering, aggregation, parsing) directly on network streams, for instance accelerating remote-memory and cache traffic before it reaches the host CPU.

4.3 GrandForge

GrandForge (illustrated Figure in Fig. 5) is the software artifact that ties the two devices to the rest of the cloud stack: a host-first library of data-intensive patterns. Rather than invent a bespoke plan format, it adopts three open building blocks. Substrait is the plan-level interoperability contract [74, 75], a portable representation of relational operations that any producer can emit and any consumer can execute; Apache Arrow [76] provides the language-independent, columnar in-memory format and, through its Acero streaming engine [77], a practical host execution substrate; and Apache Parquet [78] is the on-disk columnar layout from which data is read. The library proper is a set of composable, header-only C++ functors—kernels that consume lightweight views over Arrow-compatible buffers together with plan-derived configuration, and return batch-scoped results without owning the data or depending on runtime types. An Arrow bridge wraps each functor as a custom Acero *ExecNode*, while a Substrait reader and rewrite layer map a supported subset of relational operators onto these nodes. That subset—scan, filter, group-by/aggregate, hash join, order-by, and limit—is deliberately narrow, mirroring the executable Substrait operators so that the functor API and the plan interface stay aligned [70]. The approach generalizes the template-specialization strategy of TSL [43] from SIMD targets to near-data patterns: because operators communicate only through columnar batches and a Substrait plan, the same rewrite points double as breakpoints for offload—a device can return filtered batches, bitmap masks, or partial aggregates, while order-by and limit are handled as on-device top-k followed by a host-side merge—without altering the plan structure. The initial version of the library is host-only; device-side kernels for OpenSSD-V and OpenNIC-V are areas of active research.

5 Use Cases

The CHORYS demonstrators put the accelerator-based architecture of Section 3 to work on three use cases: a cloud object store, a data lake, and an accelerated AI-inference pipeline. The use cases are to be implemented and demonstrated in 2028. Each use case is framed around a hypothesis that near-data processing on these programmable devices reduces data movement without forfeiting the high-level abstractions cloud customers rely on. The use cases

are deliberately specified bottom-up, to illustrate that open, programmable near-processing devices, specialized through device-specific and downloadable functions, can serve very different data paths.

Cloud Object Store Object stores are the storage substrate of choice for disaggregated, cloud-native data-intensive systems, a lineage that runs from early database-on-S3 designs [79] through Snowflake [80] and Amazon Aurora [81] to today's HTAP and lakebase engines [82, 83]. As network bandwidth grows, the bottleneck shifts from raw capacity cost toward access latency and throughput [84, 85], which production systems hide behind caches. The first use case targets a Ceph deployment whose RADOS Gateway exposes an S3-compatible API over a tier of object-storage daemons, with caching at both layers—most notably the NVMe-backed D3N data cache. Recent work shows that Ceph can be re-architected for accelerator-based platforms by offloading the storage daemon to a DPU [86], and the forthcoming SeaStore back-end explicitly anticipates hardware offload [87]. Our hypothesis is that OpenNIC-V can accelerate D3N cache traffic on the network path while OpenSSD-V can limit write amplification and drive cache-tiering prefetch on the storage path in the context of a production object store. The demonstrator replays sanitized read/write traces on a rack server populated with one OpenNIC-V and several OpenSSD-V drives running the gateway and daemons.

Data Lake The second use case addresses analytics over open columnar formats. Apache Parquet [78] has become the de-facto on-disk format for data lakes thanks to its columnar layout and compression, paired with the Apache Arrow [76] in-memory representation, and it underpins the lakehouse and lakebase engines built on Delta Lake or Iceberg.

Integrating Parquet into a processing engine is, however, non-trivial: column indices and page offsets are optional or constrained by compression, distribution statistics are optional, and efficient access requires substantial host-side logic [88]. We explore improvements to Parquet index representation, that can benefit both host-based and accelerator-based processing [89].

CHORYS programs OpenSSD-V as a *Parquet SSD* that ingests Parquet files and exposes an interface tailored to the data-intensive pattern library (Section 4.3), executing hardware-accelerated sub-operators such as filtering and decompression on the device itself.

We have already carried out a feasibility study for using specialized hardware to parse Parquet files and designed a prototype implementation that exposes the different types of parallelism this task can benefit from. Even though specialized hardware cannot implement parsing logic that can handle all possible features of Parquet files, we demonstrated through an empirical study of large data sets that, in enterprise use-cases, only a small subset is actually used [90].

The hypothesis is that pushing Parquet ingestion and sub-operator evaluation into OpenSSD-V reduces data movement for a single-node OLAP worker (as opposed to streaming raw column chunks to an in-memory CPU column store [82]). The demonstrator runs a single-node OLAP benchmark over reference Parquet datasets.

Accelerated AI Inference The third use case couples a GPU running a large language model with an OpenSSD-V device to lift the limit that GPU memory places on context length. Transformer inference [91] retains a key-value (KV) cache whose size bounds the usable context; when it cannot hold the full history, output quality degrades [92], and very long contexts remain largely the preserve of hyperscalers. Because the KV cache is accessed layer by layer rather than uniformly, it can be paged between GPU memory and a programmable SSD. Here OpenSSD-V is not a passive NVMe target but an active participant on the data path: it exposes an application-specific command set, receives the per-layer KV-cache addresses and a static execution map up front, and then follows a pre-declared plan—on each per-layer *doorbell* from the GPU it pages in the next layer's KV cache and evicts the previous one, transferring directly into GPU memory through GPUDirect-style paths that bypass the CPU and host memory. This is the sharpest expression of the architecture's premise (Section 3): a storage device equipped with RISC-V cores and data-movement hardware that executes on the data path rather than merely serving blocks. The hypothesis is that such paging frees a substantial fraction of GPU memory and enables otherwise-unattainable context lengths on a single device; it requires three OpenSSD-V capabilities—persistent execution contexts, event-driven execution, and sustained low-latency transfers into GPU memory. The demonstrator evaluates a code-generation model and dataset obtained from a public model hub.

6 Conclusion

The adoption of disaggregated architectures makes near-data processing a structural necessity. CHORYS pursues it with open, programmable accelerators built around RISC-V cores and embedded directly in storage and network devices, so that data-intensive operations execute on the data path. We have set out this vision in three parts: an architecture that introduces OpenSSD-V and OpenNIC-V as drop-in devices within a conventional cloud rack and exposes them through software and management stacks that mirror

a production cloud; the artifacts that realise it—the two devices together with GrandForge, a host-first library of composable data-intensive patterns; and three use cases—an object store, a data lake, and an AI-inference pipeline—that exercise different data paths from a common substrate. The devices, their core-accelerator couplings, and the device-side patterns remain under active development. CHORYS aims to make near-data processing part of the disaggregated cloud, and to contribute that angle to the European effort to develop a reference cloud architecture.

Acknowledgements Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Health and Digital Executive Agency. Neither the European Union nor the granting authority can be held responsible for them.

Author Contributions All authors contributed equally to the manuscript.

Funding HORIZON EUROPE Global Challenges and European Industrial Competitiveness (101189551).

Data availability All data supporting the findings of this work are included in the article.

Open Access Dieser Artikel wird unter der Creative Commons Namensnennung 4.0 International Lizenz veröffentlicht, welche die Nutzung, Vervielfältigung, Bearbeitung, Verbreitung und Wiedergabe in jeglichem Medium und Format erlaubt, sofern Sie den/die ursprünglichen Autor(en) und die Quelle ordnungsgemäß nennen, einen Link zur Creative Commons Lizenz beifügen und angeben, ob Änderungen vorgenommen wurden. Die in diesem Artikel enthaltenen Bilder und sonstiges Drittmaterial unterliegen ebenfalls der genannten Creative Commons Lizenz, sofern sich aus der Abbildungslegende nichts anderes ergibt. Sofern das betreffende Material nicht unter der genannten Creative Commons Lizenz steht und die betreffende Handlung nicht nach gesetzlichen Vorschriften erlaubt ist, ist für die oben aufgeführten Weiterverwendungen des Materials die Einwilligung des jeweiligen Rechteinhabers einzuholen. Weitere Details zur Lizenz entnehmen Sie bitte der Lizenzinformation auf http://creativecommons.org/licenses/by/4.0/deed.de.

References

1. Saltzer JH, Kaashoek F (2009) Principles of Computer System Design: An Introduction. Morgan Kaufmann, Burlington, MA, USA https://doi.org/10.1016/C2009-0-20124-3
2. Barroso LA, Hölzle U, Ranganathan P (2019) The Datacenter as a Computer: Designing Warehouse-Scale Machines. Springer, Cham, Switzerland
3. Dally WJ, Turakhia Y, Han S (2020) Domain-specific hardware accelerators. Communications of the ACM 63(7):48–57
4. Kleppmann M (2017) Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems, 1st edn edn. O'Reilly Media, Sebastopol, CA, USA
5. Lerner A, Bonnet P (2024) Principles of Database and Solid-State Drive Co-Design. Synthesis Lectures on Data Management. Springer, Berlin, Germany
6. Caulfield AM, Chung ES, Putnam A, Angepat H, Fowers J, Haselman M, Heil S et al (2016) A cloud-scale acceleration architecture. 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). https://doi.org/10.1109/MICRO.2016.7783710
7. Microsoft: Project Brainwave. https://www.microsoft.com/en-us/research/project/project-brainwave/ . Accessed: 2024
8. Firestone, D., Putnam, A., Mundkur, S., Chiou, D., Dabagh, A., Andrewartha, M., Angepat, H., et al. (2018) Azure accelerated networking: SmartNICs in the public cloud. In: 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18), pp. 51–66
9. Singhvi, A., Akella, A., Gibson, D., Wenisch, T.F., Wong-Chan, M., Clark, S., Martin, M.M.K., et al. (2020) 1RMA: Re-envisioning remote memory access for multi-tenant datacenters. In: Proceedings of the ACM SIGCOMM 2020 Conference. ACM, New York, NY, USA
10. Armenatzoglou N, Basu S, Bhanoori N, Cai M, Chainani N, Chinta K, Govindaraju V et al (2022) Amazon Redshift re-invented. Proceedings of the 2022 International Conference on Management of Data (SIGMOD '22). https://doi.org/10.1145/3514221.3526045
11. Cao, W., Liu, Y., Cheng, Z., Zheng, N., Li, W., Wu, W., Ouyang, L., et al. (2020) POLARDB meets computational storage: Efficiently support analytical workloads in cloud-native relational database. In: 18th USENIX Conference on File and Storage Technologies (FAST 20)
12. Li, Y., Miao, R., Liu, H.H., Zhuang, Y., Feng, F., Tang, L., Cao, Z., et al. (2019) HPCC: High precision congestion control. In: Proceedings of the ACM SIGCOMM 2019 Conference, pp. 44–58. ACM, New York, NY, USA
13. VMware (2020) Announcing Project Monterey: Redefining Hybrid Cloud Architecture. https://blogs.vmware.com/vsphere/2020/09/announcing-project-monterey-redefining-hybrid-cloud-architecture.html
14. Intel: Why You Have Been Thinking About the Future of the Data Center All Wrong. The Six Five Summit. https://thesixfivesummit.com/session/why-you-have-been-thinking-about-the-future-of-the-data-center-all-wrong/
15. Wong, M., Himelstein, M., Ouvrement, J.: Accelerator Features for Datacenter/Cloud Computing. RISC-V Alliance white paper (in progress). https://docs.google.com/document/d/14f_ZYVGovA_xlSrja6RF846vNGnK2M0JzBn5iukdXvg/view
16. Ibanez, S., Mallery, A., Arslan, S., Jepsen, T., Shahbaz, M., Kim, C., McKeown, N. (2021) The nanoPU: A nanosecond network stack for datacenters. In: 15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21), pp. 239–256
17. Perotti, M., Cavalcante, M., Wistoff, N., Andri, R., Cavigelli, L., Benini, L. (2022) A 'new ara' for vector computing: An open source highly efficient RISC-V V 1.0 vector processor design. In: 2022 IEEE 33rd International Conference on Application-Specific Systems, Architectures and Processors (ASAP), pp. 43–51. IEEE, Piscataway, NJ, USA
18. Andes Technology: AndesCore RISC-V AX45MPV. https://www.andestech.com/en/products-solutions/andescore-processors/riscv-ax45mpv/
19. Intel: Scalable I/O Between Accelerators and Host Processors. https://www.intel.com/content/www/us/en/developer/articles/technical/scalable-io-between-accelerators-host-processors.html
20. Intel: Optimize Data Manipulation with QPL and DML. https://www.intel.com/content/www/us/en/developer/articles/technical/optimize-data-manipulation-with-qpl-and-dml.html
21. Dell'Oro Group: Smart NIC Market to Exceed 5 Billion by 2028. https://www.delloro.com/news/smart-nic-market-to-exceed-5-billion-by-2028
22. Do J, Sengupta S, Swanson S (2019) Programmable solid-state storage in future cloud datacenters. Communications of the ACM 62(6):54–62

23. TechPowerUp: Samsung PM9A3 3.8 TB SSD Specifications. https://www.techpowerup.com/ssd-specs/samsung-pm9a3-3-8-tb.d1722
24. Samsung: Samsung PM9A3 NVMe PCIe SSD. Product brochure. https://download.semiconductor.samsung.com/resources/brochure/Samsung-PM9A3-NVMe-PCIe-SSD.pdf
25. Arm: Cortex-R8 Processor. https://developer.arm.com/Processors/Cortex-R8
26. Google (2021) Aligning Storage Management with Workload Requirements. https://www.youtube.com/watch?v=3O3zDrpt3uM
27. NVM Express (2023) NVM Express Announces the Release of the Computational Storage Feature. https://nvmexpress.org/nvm-express-announces-the-release-of-the-computational-storage-feature/
28. Maschi, F., Alonso, G. (2023) The difficult balance between modern hardware and conventional CPUs. In: Proceedings of the 19th International Workshop on Data Management on New Hardware (DaMoN '23), pp. 53–62. ACM, New York, NY, USA
29. Kwak J, Lee S, Park K, Jeong J, Song YH (2020) Cosmos+ OpenSSD: Rapid prototype for flash storage systems. ACM Transactions on Storage. https://doi.org/10.1145/3385073
30. NetFPGA: NetFPGA. https://netfpga.org/
31. T-Head: Zhenyue 510 Enterprise SSD Controller. https://www.t-head.cn/product/Zhenyue510
32. Phison: X1 Enterprise PCIe SSD. https://www.phison.com/en/solutions/enterprise/pcie/x1-ssd
33. Lee, S., Lerner, A., Bonnet, P., Cudré -Mauroux, P. (2024) Database kernels: Seamless integration of database systems and fast storage via CXL. In: 14th Conference on Innovative Data Systems Research (CIDR '24)
34. Jeong, J., Park, K., Lee, S., Bonnet, P., Lerner, A., Cudré-Mauroux, P. (2023) Far-and-near: Co-designed storage reliability between database and SSDs. In: 13th Conference on Innovative Data Systems Research (CIDR '23)
35. AMD/Xilinx: OpenNIC. https://github.com/Xilinx/open-nic
36. Forencich, A.: Corundum: An Open-Source FPGA-Based NIC. https://github.com/corundum/corundum
37. AMD/Xilinx: Vitis Networking P4 (VitisNetP4). https://www.xilinx.com/products/intellectual-property/ef-di-vitisnetp4.html
38. NVIDIA: DOCA Software Framework. https://developer.nvidia.com/networking/doca
39. Giouroukis D, Nugroho DPA, Pandey V, Zeuch S, Markl V (2025) Analyzing near-network hardware acceleration with co-processing on DPUs. Proceedings VLDB Endowment 18(13):5689–5702. https://doi.org/10.14778/3773731.3773743
40. Khronos Group: SYCL. https://www.khronos.org/sycl/
41. UXL Foundation: oneAPI. https://uxlfoundation.org/
42. Shilpage WR, Wright SA (2023) An investigation into the performance and portability of SYCL compiler implementations. In: High Performance Computing. Springer, Cham, Switzerland, pp 605–619
43. TU Dresden Database Systems Group: TSL: Template SIMD Library. https://github.com/db-tu-dresden/TSL
44. Lerner, A. (2025) Initial system architecture. Deliverable D2.1, CHORYS Consortium. https://doi.org/10.5281/zenodo.17908243
45. Ghemawat S, Gobioff H, Leung ST (2003) The Google file system. Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP '03). https://doi.org/10.1145/945445.945450
46. Burrows, M. (2006) The Chubby lock service for loosely-coupled distributed systems. In: Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI '06), pp. 335–350. USENIX Association, Berkeley, CA, USA
47. Wang, T., Yang, X., Antichi, G., Sivaraman, A., Panda, A. (2022) Isolation mechanisms for high-speed packet-processing pipelines. In: 19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22), pp. 1289–1305. USENIX Association, Berkeley, CA, USA
48. Armbrust M, Ghodsi A, Xin R, Zaharia M (2021) Lakehouse: A new generation of open platforms that unify data warehousing and advanced analytics. 11th Conference on Innovative Data Systems Research (CIDR '21) 8:28
49. Pal KK, Sudeep KS (2016) Preprocessing for image classification by convolutional neural networks. 2016 IEEE International Conference on Recent Trends in Electronics, Information and Communication Technology (RTEICT). https://doi.org/10.1109/RTEICT.2016.7808140
50. Verma A, Pedrosa L, Korupolu M, Oppenheimer D, Tune E, Wilkes J (2015) Large-scale cluster management at Google with Borg. Proceedings of the 10th European Conference on Computer Systems (EuroSys '15). https://doi.org/10.1145/2741948.2741964
51. Tirmazi M, Barker A, Deng N, Haque ME, Qin ZG, Hand S, Harchol-Balter M, Wilkes J (2020) Borg: The next generation. Proceedings of the 15th European Conference on Computer Systems (EuroSys '20). https://doi.org/10.1145/3342195.3387517
52. Sigelman BH, Barroso LA, Burrows M, Stephenson P, Plakal M, Beaver D, Jaspan S, Shanbhag C (2010) Dapper, a large-scale distributed systems tracing infrastructure. (Technical report, Google Inc)
53. Beyer B, Jones C, Petoff J, Murphy NR (eds) (2016) Site Reliability Engineering: How Google Runs Production Systems. O'Reilly Media, Sebastopol, CA, USA
54. Kestor G, Gioiosa R, Kerbyson DJ, Hoisie A (2013) Quantifying the energy cost of data movement in scientific applications. 2013 IEEE International Symposium on Workload Characterization (IISWC). https://doi.org/10.1109/IISWC.2013.6704670
55. Ports DRK, Nelson J (2019) When should the network be the computer? Proceedings of the Workshop on Hot Topics in Operating Systems (HotOS '19). https://doi.org/10.1145/3317550.3321439
56. Thompson NC, Spanuth S (2021) The decline of computers as a general purpose technology. Communications of the ACM 64(3):64–72. https://doi.org/10.1145/3430936
57. Batcher KE (1968) Sorting networks and their applications. Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference (AFIPS '68 (Spring)). https://doi.org/10.1145/1468075.1468121
58. Abadi DJ, Madden SR, Hachem N (2008) Column-stores vs. row-stores: How different are they really? Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data (SIGMOD '08). https://doi.org/10.1145/1376616.1376712
59. Caminal H, Chronis Y, Wu T, Patel JM, Martínez JF (2022) Accelerating database analytic query workloads using an associative processor. Proceedings of the 49th Annual International Symposium on Computer Architecture (ISCA '22). https://doi.org/10.1145/3470496.3527435
60. Purayil NK, Perotti M, Fischer T, Benini L (2025) AraXL: A physically scalable, ultra-wide RISC-V vector processor design for fast and efficient computation on long vectors. https://arxiv.org/abs/2501.10301 (arXiv preprint arXiv:2501.10301)
61. Kocberber O, Grot B, Picorel J, Falsafi B, Lim K, Ranganathan P (2013) Meet the walkers: Accelerating index traversals for in-memory databases. Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-46). https://doi.org/10.1145/2540708.2540748
62. Cammarata D, Perotti M, Bertuletti M, Garofalo A, Schiavone PD, Atienza D, Benini L (2025) Quadrilatero: A RISC-V programmable matrix coprocessor for low-power edge applications.
63. Kung HT, Leiserson CE (1979) Systolic arrays (for VLSI). In: Sparse Matrix Proceedings 1978, vol 1. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, pp 256–282
64. Schuiki F, Zaruba F, Hoefler T, Benini L (2021) Stream semantic registers: A lightweight RISC-V ISA extension achieving full com-

pute utilization in single-issue cores. IEEE Transactions on Computers 70(2):212–227. https://doi.org/10.1109/TC.2020.2987314
65. Guo, L., Zuberi, D., Garfinkel, T., Ousterhout, A. (2025) The benefits and limitations of user interrupts for preemptive userspace scheduling. In: 22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25), pp. 1015–1032. USENIX Association, Berkeley, CA, USA
66. Asanović K, Lee Y, Avizienis R, Richards B, Waterman A, Cook H, Morgan Z, Patterson DA (2016) The Rocket chip generator. (Technical Report UCB/EECS-2016-17, EECS Department, University of California, Berkeley)
67. Damian M, Oppermann J, Spang C, Koch A (2022) SCAIE-V: An open-source scalable interface for ISA extensions for RISC-V processors. Proceedings of the 59th ACM/IEEE Design Automation Conference (DAC '22). https://doi.org/10.1145/3489517.3530432
68. OpenHW Group (2024) Core-V eXtension Interface (CV-X-IF) Specification. (OpenHW Group, Version 1.0.0-rc.3-dev.3)
69. Bonnet P, Lerner A, Zoni D, Antichi G, Pietrzyk J (2025) Gap analysis and requirements. Deliverable D2.1, CHORYS Consortium. https://zenodo.org/communities/chorys (Horizon Europe project CHORYS (Grant Agreement No. 101189551))
70. Pietrzyk J, Bonnet P, István Z, Habich D (2025) Library of host-based data-intensive patterns. Deliverable D7.1, CHORYS Consortium. https://github.com/db-tu-dresden/chorys (Horizon Europe project CHORYS (Grant Agreement No. 101189551))
71. Lerner, A. (2026) Initial fpga-based platforms for co-designed ssds and nics. Deliverable D3.2, CHORYS Consortium. https://doi.org/10.5281/zenodo.22046120
72. Haas, G., Lee, B., Bonnet, P., Leis, V. (2025) SSD-iq: Uncovering the hidden side of SSD performance. Proceedings VLDB Endowment 18(11), 4295–4308. https://doi.org/10.14778/3749646.3749694
73. Zaruba F, Schuiki F, Hoefler T, Benini L (2021) Snitch: A tiny pseudo dual-issue processor for area and energy efficient execution of floating-point intensive workloads. IEEE Transactions on Computers 70(11):1845–1860
74. Nadeau, J. (2022) Substrait: Rethinking DBMS composability. In: 1st International Workshop on Composable Data Management Systems (CDMS@VLDB 2022), Sydney, Australia
75. (2025) Substrait: Cross-Language Serialization for Relational Algebra. https://substrait.io. Accessed 22 Dec 2025
76. Apache Software Foundation: Apache Arrow. https://arrow.apache.org
77. Apache Software Foundation: Apache Acero. https://arrow.apache.org/docs/cpp/acero/overview.html.
78. Apache Software Foundation: Apache Parquet. https://parquet.apache.org
79. Brantner, M., Florescu, D., Graf, D., Kossmann, D., Kraska, T. (2008) Building a database on S3. In: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data. ACM, New York, NY, USA
80. Dageville, B., Cruanes, T., Zukowski, M., Antonov, V., Avanes, A., Bock, J., Claybaugh, J., Engovatov, D., Hentschel, M., Huang, J., Lee, A.W., Motivala, A., Munir, A.Q., Pelley, S., Povinec, P., Rahn, G., Triantafyllis, S., Unterbrunner, P. (2016) The snowflake elastic data warehouse. In: Proceedings of the 2016 International Conference on Management of Data (SIGMOD '16). ACM, New York, NY, USA
81. Verbitski, A., Gupta, A., Saha, D., Brahmadesam, M., Gupta, K., Mittal, R., Krishnamurthy, S., Maurice, S., Kharatishvili, T., Bao, X. (2017) Amazon aurora: Design considerations for high throughput cloud-native relational databases. In: Proceedings of the 2017 ACM International Conference on Management of Data (SIGMOD '17). ACM, New York, NY, USA
82. Schmidt T, Durner D, Leis V, Neumann T (2024) Two birds with one stone: Designing a hybrid cloud storage engine for HTAP. Proceedings of the VLDB Endowment 17(11):3290–3303
83. Zaharia M (2025) Bringing the operational and analytical worlds together with lakebase. Proceedings of the VLDB Endowment 18(12):5539
84. Durner D, Leis V, Neumann T (2023) Exploiting cloud object storage for high-performance analytics. Proceedings of the VLDB Endowment 16(11):2769–2782
85. Warfield, A. (2023) Building and operating a pretty big storage system (my adventures in amazon S3). In: 21st USENIX Conference on File and Storage Technologies (FAST 23). USENIX Association, Santa Clara, CA, USA
86. Park, K., Yoon, S., Talibli, F., Park, S., Kwak, J.-H., Jeong, K., Khan, A., Kim, Y. (2025) DoCeph: DPU-offloaded messaging in ceph for reduced host CPU utilization. In: Proceedings of the SC '25 Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 1658–1666. ACM, New York, NY, USA
87. Just S (2022) SeaStore: Next Generation Backing Store for Ceph. Ceph developer presentation.
88. Rey, A., Freitag, M., Neumann, T. (2023) Seamless integration of parquet files into data processing. In: BTW 2023, pp. 235–258. Gesellschaft für Informatik e.V., ???
89. Faghih, F., Kwon, S.J., Istv´an, Z. (2026) What no one tells you about page pruning in Parquet: The real cost of page index parsing. In: Proceedings of the 1st International Workshop on Data Formats for Modern Architectures and Workloads (FORMATS). Co-located with ACM SIGMOD/PODS 2026, Bengaluru, India
90. Kwon, S.J., Istv´an, Z., Ritter, D., May, N., Faerber, C. (2026) Efficient parquet parsing on FPGAs. In: Proceedings of the International Workshop on Data Management on New Hardware (DaMoN). Co-located with SIGMOD/PODS 2026
91. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I. (2017) Attention is all you need. In: Advances in Neural Information Processing Systems 30 (NeurIPS 2017)
92. Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C.H., Gonzalez, J.E., Zhang, H., Stoica, I. (2023) Efficient memory management for large language model serving with PagedAttention. In: Proceedings of the 29th Symposium on Operating Systems Principles (SOSP '23), pp. 611–626. ACM, ???

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.