

# AMNES: Accelerating the computation of data correlation using FPGAs

Monica Chiosa Systems Group, Dept. of Computer Science, ETH Zurich monica.chiosa@inf.ethz.ch

> Michaela Blott AMD Research and Advanced Development michaela.blott@amd.com

# ABSTRACT

A widely used approach to characterize input data in both databases and ML is computing the correlation between attributes. The operation is supported by all major database engines and ML platforms. However, it is an expensive operation as the number of attributes involved grows. To address the issue, in this paper we introduce AMNES, a stream analytics system offloading the correlation operator into an FPGA-based network interface card. AMNES processes data at network line rate and the design can be used in combination with smart storage or SmartNICs to implement near data or in-network data processing. AMNES design goes beyond matrix multiplication and offers a customized solution for correlation computation bypassing the CPU. Our experiments show that AMNES can sustain streams arriving at 100 Gbps over an RDMA network, while requiring only ten milliseconds to compute the correlation coefficients among 64 streams, an order of magnitude better than competing CPU or GPU designs.

#### **PVLDB Reference Format:**

Monica Chiosa, Thomas B. Preußer, Michaela Blott, and Gustavo Alonso. AMNES: Accelerating the computation of data correlation using FPGAs. PVLDB, 16(13): 4174 - 4187, 2023. doi:10.14778/3625054.3625056

#### **PVLDB Artifact Availability:**

Source code available at https://github.com/fpgasystems/amnes.

# **1** INTRODUCTION

Correlation is a term frequently used in machine learning [27], data mining [16], databases [35, 51], business analysis [52], and statistics [70] to interchangeably represent different types of relations (linear or non-linear), mutual dependencies, or causality, with the ultimate goal of summarizing large amounts of data by observing patterns between variables [8]. Correlation is important for data processing and data management systems [11, 35, 66] and usually finds its place in exploration, data cleaning, or data pre-processing stages, all constituting a significant effort for a data scientist [45]. For example, knowing the correlation among data can affect the scheduling decision of whether to offload the computation to a General Purpose GPU (GPGPU) or another accelerator that involves Thomas B. Preußer

AMD Research and Advanced Development thomas.preusser@amd.com

Gustavo Alonso

Systems Group, Dept. of Computer Science, ETH Zurich alonso@inf.ethz.ch

data movement [9]. Similarly, correlated data affects the error of the selectivity estimators used by query optimizers [19], i.e., highly correlated data leading to higher errors [28]. In machine learning, knowing which features or dimensions are correlated serves in both dimensionality reduction, by pointing at data that can be removed as it does not provide additional information [81], and correlation clustering since the correlation between data is a useful input to clustering algorithms [40]. Finally, correlation computation can be utilized by vector databases to find data vector similarity [57], or in data privacy applications to detect when too much public data leads to information leakage [84].

Finding the correlation between data sets (columns in a relational table, two data streams, or dimensions in a set of vector data) is typically an expensive operation. On the one hand, its computation typically involves calculating several statistics over each data set that feed into computing the correlation coefficient. This requires a full pass over the data. On the other hand, in most cases, the correlation has to be calculated across many data sets (e.g., wide tables in databases, or high-dimensional vectors in ML applications). As a reference, the widely used NumPy and Pandas Python libraries can take up to 100 ms to compute the correlation between just 16 variables with 200'000 elements each. Later in the paper, we show how to compute the correlation for 64 streams and 2 million elements per variable in about 10 ms, i.e., an order of magnitude less than these established libraries. In databases, due to the cost of computing correlation, it is often approximated, especially when used to optimize the creation of indexes, as data correlations can significantly affect their performance, particularly for clustered indexes [17, 39]. One common approximation is to compare the number of distinct values across attributes [39]. However, this measure is far less precise than statistical correlation measures.

Relational engines such as Postgres, Oracle, MySQL, or Snowflake and vector databases (Milvus) compute correlation between pairs of attributes via either an intrinsic operator (Postgres, Oracle DB, Snowflake, BigTable) or by combining data statistics (MySQL). For more than two attributes, a manual query with the explicit pairs of attributes has to be written. As the amount of data to be processed grows, there is a need to understand how to efficiently correlate many attributes in parallel and whether the computation can be offloaded to an accelerator, e.g., without CPU involvement. In this paper we explore this question by looking at how computing the statistical correlation can be accelerated using an FPGA acting on streams of data arriving from or being sent to the network.

The reason to explore such a design is based on the fact that storage is nowadays often disaggregated with data processing involving first reading the data from object storage and bringing it into the

This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit https://creativecommons.org/licenses/by-nc-nd/4.0/ to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.

Proceedings of the VLDB Endowment, Vol. 16, No. 13 ISSN 2150-8097. doi:10.14778/3625054.3625056

computing node. This opens up the possibility to offload computation to accelerators either on the storage nodes or on the network path. Examples of such systems include: (1) Amazon AQUA, which employs FPGAs together with SSDs to offload SQL operators (selection, projection, LIKE predicates, etc.) to a network-attached caching layer [7]; (2) Microsoft's Catapult, which has an FPGA along the network data path to accelerate a wide range of use cases from key-value stores [47], network function virtualization [22], search engines [53], to AI/ML applications [49]; and (3) Oracle Exadata, a database engine with smart disaggregated storage where data is kept in row format for online transaction processing (OLTP), but is transformed on the fly into column-based as data is moved from storage to in-memory to accommodate fast online analytical processing (OLAP). In all these cases, our design enables offloading the correlation operator to the accelerator or to the storage.

These are the scenarios targeted by AMNES although, as we show in the paper, it can also be used in the conventional acceleration model with data residing in the CPU's memory. The benefit of having the correlation assessed while data is moving through the network, be it from one compute node to another or from storage to a compute node, is that correlation becomes meta-data that can be used by the subsequent processing tasks (e.g., machine learning pipelines, scheduling algorithms, or analytical jobs), once all the transferred data has reached its destination. Since most of the data to be processed in cloud-based systems has to travel through the network, enhancing network cards with complex computation capabilities reduces data movement, eliminates the need for intermediate storage, and eventually reduces energy consumption.

To show the potential of these ideas, we have developed AMNES, an open source [75] FPGA-based accelerator that computes the Pearson correlation coefficient (PCC) among data streams. AMNES can operate on data residing in host memory (i.e., as a conventional accelerator due to its low-latency) or on data streams arriving from the network. While the actual network transport protocol used is not relevant for the computation, in our prototype, we have focused on RDMA networks due to their higher throughout challenges and significantly lower data access latency than TCP/IP. AMNES operates specifically on the Converged Ethernet (RoCEv2) protocol, which is already deployed by many cloud providers such as Alibaba Cloud [24], Microsoft Azure [26], with 70% of Azure traffic being RDMA-based [54], and Oracle Exadata [59]. RDMA has also made its way into database design, with increasing systems and prototypes demonstrating its advantages [55, 82].

Finally, we note that we focus on Pearson's correlation as it is the most widely used. Kendall and Spearman correlations can be easily computed using a simplified variation of our design to process sorted data (as these two correlation operators required the data to be sorted). AMNES's modular architecture allows to use only part of the design (e.g., the ACC Engine), to derive a variety of useful statistics such as correlation coefficients, cosine similarity and cosine distance, standard deviation, as well as slope and intercept of linear regression lines over the data. These are all standard operators in relational engines these days and can be supported by our design with minimal changes.

Through AMNES design we make the following contributions: (1) the description of an FPGA-based accelerator computing Pearson

correlation coefficients for parallel data streams; (2) the demonstration of the concurrent computation of the correlation among up to 64 data streams and the analytical study of the capacity of the design, which is as high as several thousand concurrent data streams, well beyond the I/O capabilities of modern devices or networks; (3) the demonstration of the ability to embed the accelerator on an RDMA-capable SmartNIC; and (4) the evaluation of the design proving that it offers several orders of magnitude performance gains in both throughput and latency over CPU and GPU-based approaches.

## 2 BACKGROUND

In this section, we formally define the correlation between two streams and introduce the FPGA. Furthermore, we explain the statistical and mathematical background underlying the Pearson correlation coefficient and present our motivation behind using RDMA (Remote Direct Memory Access) at networking level.

## 2.1 The Correlation Coefficients

Correlation can be measured using different types of coefficients, e.g., Pearson, Kendall, Spearman, or Point-Biserial. The latter employs the same formula as the Pearson correlation coefficient (PCC), with one variable being binary. The Pearson correlation coefficient is the normalized version of covariance [16]. In contrast to Pearson, which measures linear association, Kendall and Spearman coefficients are non-parametric tests (e.g., do not depend on the underlying data distribution) and measure an ordinal association [71] between ranked data. These two coefficients assume ordered stream values and are mainly used to assess a non-linear association [20]. Database engines often include several measures of correlation as part of their statistical functions (e.g., Oracle includes support for Pearson, Kendall, and Spearman correlation coefficients).

In this paper, we focus on the Pearson correlation coefficient, which measures the strength of the linear association between two data streams by taking into account the amount of variation present in each stream and how the streams vary together. The coefficient is a dimensionless quantity within the range [-1, +1]. Unlike covariance, which can take infinite values, this well-confined range allows for a straightforward assessment. A Pearson correlation coefficient  $(\rho)$  of value 0 indicates that no linear relationship exists between two data streams, i.e., they are independent. A perfect linear relationship is indicated by a coefficient of magnitude 1. A negative sign indicates that the increase in the values of one stream associates with a decrease in the values of the other stream. A positive sign, on the contrary, indicates that the increase in the values of one stream associates with the increase in the values of the other. The stronger the correlation, the closer the correlation coefficient gets to  $\pm 1$ . If we consider  $|\rho|$ , the absolute value of the PCC, the strength of the linear relationship can be assessed as follows: (1) weak correlation for  $|\rho| \in [0.1, 0.3)$ ; (2) medium correlation for  $|\rho| \in [0.3, 0.5)$ ; and (3) strong correlation for  $|\rho| \in [0.5, 1]$  [6, 72]. Nevertheless, the relationship strength does not imply any causal relationship between the two given streams [1].

The PCC is the most widely used among the correlation coefficients presented above [71]. Intuitively, if a line is drawn as a best fit through the data points of two streams, the PCC indicates the amount of variation that exists around this line of best fit. Nevertheless, PCC does not represent the slope of the line of best fit. Its value in [-1,+1] indicates the variation around this line, with values closer to 0 indicating a large variation.

Mathematically, the PCC ( $\rho$ ) for a bound population (e.g., maximum number of items for two analyzed streams is N) is shown in Equation 1, where  $\mathbb{E}$  is the expectation,  $\mu$  and  $\sigma$  are the mean and the standard deviation of each data stream. Developed further, Equation 1 describes PCC as the centered and standardized sum of the cross-product of two data streams [63].

$$\rho_{XY} = \frac{\mathbb{E}[(X - \mu_X)(Y - \mu_Y)]}{\sigma_X \sigma_Y} \\ = \frac{\sum_{t=0}^{N-1} (x_t - \mu_X)(y_t - \mu_Y)}{\sqrt{\sum_{t=0}^{N-1} (x_t - \mu_X)^2} \sqrt{\sum_{t=0}^{N-1} (y_t - \mu_Y)^2}}$$
(1)

In Equation 2, we replace expectation and standard deviation by their corresponding mathematical formulas and reduce the common members to minimize the number of necessary divisions. Equation 2 leads to the core components around which our design centers: the sum of elements  $(\sum_{t=0}^{N-1} x_t, \sum_{t=0}^{N-1} y_t)$ , the sum of squares  $(\sum_{t=0}^{N-1} x_t^2, \sum_{t=0}^{N-1} y_t^2)$ , and the sum of products  $(\sum_{t=0}^{N-1} x_t y_t)$ . For the remaining of the paper, we are going to refer to them as *the sufficient statistics*. A similar equation is derived for binary variables and used to find the correlation between graphs [37], aiming at discovering the dependencies within a graph database.

$$\rho = \frac{\sum_{t=0}^{N-1} x_t y_t - \frac{\sum_{t=0}^{N-1} x_t \sum_{t=0}^{N-1} y_t}{N}}{\sqrt{\sum_{t=0}^{N-1} x_t^2 - \frac{(\sum_{t=0}^{N-1} x_t)^2}{N}} \sqrt{\sum_{t=0}^{N-1} y_t^2 - \frac{(\sum_{t=0}^{N-1} y_t)^2}{N}}} = \frac{N \sum_{t=0}^{N-1} x_t y_t - \sum_{t=0}^{N-1} x_t \sum_{t=0}^{N-1} y_t}{\sqrt{N \sum_{t=0}^{N-1} x_t^2 - (\sum_{t=0}^{N-1} x_t)^2} \sqrt{N \sum_{t=0}^{N-1} y_t^2 - (\sum_{t=0}^{N-1} y_t)^2}}$$
(2)

In order for the PCC results to be interpretable and trusted, the analyzed streams should satisfy the following assumptions [72]: (1) no missing values and a continuous scale; (2) the stream values should be normally distributed, have a linear relationship and same variance around the regression line; and (3) the streams should not have outliers, values that do not follow a similar pattern as the rest of the data. Moreover, PCC is applicable only to numeric values [8]. The correlation involving data of other types, such as strings, requires their mapping to numeric values.

In realistic use cases, it is unavoidable for the data to not have missing values, non-normal distribution or outliers. There are several ways to overcome this and still have an interpretable PCC value: (1) pairwise missing values - compute the correlation using the non-missing streams' values; this results in a partial correlation coefficient [8]; (2) list-wise deletion - compute the correlation only using observations with non-missing values for both streams [8, 21]; (3) replace the missing values by either means among the adjacent values or by constants. Each of these approaches are easy to implement on an FPGA, at the cost of one clock cycle increase in latency. In the design we explore in this paper, we have not included this feature as it has no impact on the overall result. To assess the normality of stream value distribution, one can employ either histograms [32] or the Jarque-Bera test [34], with the former being successfully implemented on FPGAs [32]. Various techniques exist for outlier detection, ranging from statistical methods such as Z-score and Mahalanobis distance to machine learning approaches such as clustering or support vector machines. However, for FPGA implementations aiming to maintain 100 Gbps rates, moving average or exponential smoothing techniques are more appropriate [33].

Related Work. In databases, correlation has been used interchangeably to capture three different concepts. The first is the semantic relationship (e.g., a functional dependency) between columns. Hermit [77], Correlation Maps [38], CORDS [30], BHUNT [10], CORADD [39] use attributes' semantic correlation to improve indexing, query execution, and query optimization [51] performance. The second concept uses the two attributes covariance to model selectivity for query optimizers [14]. And the third models the relationship between pairs in time series analysis: BRAID [66] for lag correlation; StatStream [85], Mueen et al. [56], and Li et al. [48] for longest-lasting correlated subsequences; and Wadjet [65] for outlier identification. The last two concepts employ PCC in a CPU based streaming context. As noted, correlation computation on the CPU is expensive and is typically approximated or limited to specific data segments (i.e., Xiong et al. [80] opt to focus on Zipfian distributed datasets for large number of streams and compute PCC for only a subset of pairs; the same applies for Zhang and Feigenbaum [83] to find correlation among large datasets).

Correlation on Heterogeneous Architectures. FPGA-based correlation implementations have been proposed for image processing [41], OFDM (orthogonal frequency division multiplexing) timing synchronization [60], and digital correlation processors [4]. Image correlation differs from PCC by evaluating pixel value and energy differences between images, but shares similarities in its use of additions and multiplications for computation. Nevertheless, image processing correlation uses  $11 \times 11$  window values, leading to computations over 121 pixels, considerably smaller than our target stream lengths. OFDM timing synchronization [60] aims to reduce DSP utilization by replacing multiplication operations with shift-and-adds, resulting in approximated results. In contrast, our approach focuses on analyzing classic data types, providing exact results, and preserving data representation. There are limited customized GPU-based correlation implementations due to data movement overhead and the GPU's constrained memory capacity. Chang et al. [12] achieved a significant speedup of 28× to 38× in PCC computation for matrix sizes ranging from  $4096 \times 16$  to  $12288 \times 64$ , utilizing floating-point representations for sequence database search. Their assessment considered both computation and data transfer time to and from the GPU memory. However, the GPU's memory limitations constrained the maximum analyzed size, and the execution time was in the order of seconds, falling short of the performance and architectural flexibility achieved with AMNES.

Conversely to this previous work, we focus on computing the PCC on data streams at network line rate for direct deployed on the network without CPU involvement. We show correlation computation for up to 64 parallel data streams (2016 PCC values) with several orders of magnitude performance gains over existing solutions, and how the design can be generalized for correlating thousands of streams, if sufficient bandwidth is available. This demonstrates the ongoing validity of our design as networks and CPU-accelerator (e.g., Intel CXL interconnects) bandwidths improve.

# 2.2 RDMA

In this work, we use RDMA over Converged Ethernet (RoCE v2), with the entire network stack [73] being deployed on an FPGAbased SmartNIC [43] that we test over a 100 Gbps HACC cluster [74]. On the FPGA-based SmartNIC, AMNES acts as a bump-in-the-wire accelerator, being placed between the network stack module (ensures the communication of the FPGA with the RDMA network) and the PCIe module (ensures the communication of the FPGA with the host CPU). For testing purposes, we use the low-latency, one-sided RDMA operations, namely the write primitive. The computational kernel is placed at the receiving node, and the correlation coefficients are computed as data is arriving through the network from RDMA write requests. Although not explored in this paper, the same could be done on the sending node to compute the correlation near to the data source. The same behavior is expected from AMNES if it is utilized together with the read primitive (computing correlation while data is received over the network, after sending an RDMA read request). By utilizing the read operation, the communication latency increases, since one more network trip is required before data gets sent over the network. The concept remains the same for the two one-sided RDMA operations, with the overhead being completely independent of the correlation computation.

Related Work. RDMA has gained prominence in data centers, finding applications in distributed systems, databases, cloud storage, and in-network data analytics [2, 18, 24, 44, 46, 58, 62, 64, 67, 76, 82]. StRoM [67], a system leveraging SmartNICs with RDMA support, pioneered the integration of compute capabilities into the RDMA network stack, demonstrating the computation of the Hyperloglog (HLL) cardinality approximation algorithm while data traverses the network. Similarly, we illustrate how a correlation engine can be placed on the SmartNIC without impacting network performance. Unlike StRoM, which hashes input values to dissociate input data representation from the algorithm's internal structure, AMNES retains the original data representation for processing. Farview, another FPGA-based SmartNIC system, utilizes RDMA to offload query operators to a network-attached DRAM module, achieving performance comparable to local memory [42]. Farview serves as a potential deployment example for AMNES in future data centers.

#### **3 CORRELATION ENGINE**

In this section, we present the AMNES design and focus on its two main components (Figure 1): the accumulator engine (ACC Engine) and the coefficient engine (COEFF Engine). The *pre-* and *post*processing modules prepare the data either to enter the AMNES's compute engines (i.e., augment the data with a control signal marking the last element to be analyzed) or to be sent to the host CPU (i.e., combine the coefficient results to fit into a cacheline). The BRAM



Figure 1: AMNES block diagram.

Table 1: Symbols defining basic design parameters.

Ν	Total number of elements of each stream.
M	Number of streams in a cacheline ( $M \ge 2$ ).
$G_p$	Number of unique pairs between M streams.
si	Stream belonging to a cacheline, $i \in [0, M)$ .
$s_{i_t}$	Data items of the stream $s_i$ , $t \in [0, N)$ .
LACC	Latency of the ACC Engine [clock cycles].
L <sub>COEFF</sub>	Latency of the COEFF Engine [clock cycles].
Width <sub>data</sub>	Input data representation [B].
$Width_{ACC}$	Accumulator representation [B].

temporally stores the results generated by the ACC Engine before being used by the COEFF Engine. The URAM and HBM (High Bandwidth Memory) memories are considered for design generalization.

## 3.1 System Overview

We consider as *a stream* a continuous finite flow of fixed-size data items entering AMNES's compute engine. The compute granularity of the design is at cacheline level (64 B - 512 bits) leading to multiple streams to be analyzed in parallel, with each data item in the cacheline being *a value of a certain stream*. The maximum number of streams (M) analyzed in parallel depends on the data item representation (*Width<sub>data</sub>*), i.e., 64 streams for 8 bits, 32 streams for 16 bits and 16 streams for 32 bits data items, respectively. Table 1 summarizes the symbols used in describing the design.

AMNES computes the PCC from the sufficient statistics as required by Equation 2. These comprise, for each stream  $s_i$ , the sum of elements -  $S_{e_i} = \sum_{t=0}^{N-1} s_{i_t}$  and the sum of squares -  $S_{sq_i} = \sum_{t=0}^{N-1} s_{i_t}^2$ , as well as, for each unique pair of streams, the sum of products - $S_{p_z} = \sum_{t=0}^{N-1} s_{i_t} s_{j_t}$  with  $i < j, i, j \in [0, M)$  and  $z \in [0, G_p)$ . After gathering all these statistics, the accumulated values are used to obtain the PCC between each unique streams pair. AMNES's design takes as input M streams with N data items each, and produces M(M-1)/2 correlation coefficients, one coefficient for each unique pair of distinct streams. Note that the commuted pairs  $(s_i, s_j)$  and  $(s_i, s_i)$  are not differentiated as they yield the same coefficient. As depicted in Figure 1, the design splits into two parts: the backend part (accumulators engine - ACC) and the frontend part (coefficients engine - COEFF). The backend part gathers the sufficient statistics, whereas the frontend computes the coefficient values. AMNES is implemented in C++ as a customizable streaming Vitis HLS (High Level Synthesis) kernel and deployed on FPGA as a compute kernel.

The challenge of the implementation is to obtain processing pipelines for each of the engines that guarantee an *initiation interval* of 1 (II = 1), i.e., at every clock cycle, the compute engine is capable of consuming one cacheline of data items from M parallel streams. An FPGA pipeline is similar in concept to a pipelined processor architecture, with each stage of the pipeline executing a different operation, thus enabling concurrent execution of tasks. On the FPGA, the individual stages are separated by registers. While a deeper pipeline with more stages implies a higher processing latency in terms of clock cycles, its more fine-granular segmentation into stages will typically reduce the most critical signal path and lead to a higher operational clock frequency and throughput.

Table 2: ACC resources characterization for M streams.

Data Width [bit]	Streams	Accumulators	MAC Units	AMNES Op. Freq. [MHz]		
32	16	16	136	300		
16	32	32	528	250		
8	64	64	2080	190		

#### 3.2 The ACC Engine

The backend part gathers the sufficient statistics in parallel for M streams through a network of accumulators and multiply-accumulate (MAC) units. For M streams, M accumulators are required for sum of elements- $S_e$ , and M(M + 1)/2 MAC units are required for sum of squares- $S_{sq}$  and sum of products- $S_p$  (M units for  $S_{sq}$  and M(M-1)/2units for  $S_p$ ). The number of MAC units has a quadratic dependency on the number of streams that are analyzed in parallel. Table 2 shows the number of MAC units dependent on the data types (i.e., 8-bits, 16-bits, 32-bits) we consider for our implementation. In C++, each type of sum  $(S_e, S_{sq}, S_p)$  is associated with a class that exposes a set of functions that act upon the data sent to them. Since  $S_e$ and  $S_{sq}$  can be computed independently for each stream in the cacheline, AMNES associates objects from these two classes to each of the streams of the cacheline. Sp depends on the values coming from all the streams, so only one object is associated from this class to all the streams included in a cacheline. ACC Engine latency lower bound  $(L_{ACC})$  is given by the latency of the multiply operator, e.g., 3 clock cycles. Vitis HLS might introduce a few more cycles as latency on top of this lower bound for larger number of streams (e.g., 64). The bit representation of the stream's items is customized via  $Width_{data}$  parameter, whereas the bit representation for the accumulators associated with each sum is customized via WidthACC parameters. In our implementation, the choice of  $Width_{ACC} = 2 * Width_{data}$  accommodates stream lengths of up to 2 millions items per stream, with the consideration that some values are repeated. Since the FPGA offers customizable data width representations, Table 3 analyzes maximum bit representations of supported unsigned integer representations for accumulators when no pre-processing is applied for corner cases.

FPGAs are known for their flexibility in terms of customizable widths for different data types: from integer and fixed-point to floating-point representations; with the latter being the most resource intense and slowest of the three (i.e., modules working with floating-point values have a lower operating frequency than the modules working with integer or fixed-point values). Since our focus is on relational data and potentially machine learning systems that use fixed-point or low-precision representations, our design centers on integer and fixed-point representations.

## 3.3 The COEFF Engine

The COEFF Engine takes all the sufficient statistics previously gathered and generates  $G_p$  floating-point values representing the Pearson correlation coefficient  $(c_{ij})$  between the unique stream pairs,  $(s_i, s_j)$ , with i < j and  $i, j \in [0, M)$ . The frontend computation is triggered once all the input data from the M streams has been consumed by the ACC Engine. When multiple streams (attributes) are correlated, a square matrix of correlation coefficients outputs is usually presented as in Figure 2a. This matrix exhibits a diagonal line consisting of '1' values, representing the correlation of

Table 3: *Width<sub>ACC</sub>* analysis.

Data Width [bit]	All values	Width <sub>ACC</sub> for 1 million [bit]	Width <sub>ACC</sub> for 2 millions [bit]
32	1	21	22
	$2^{32} - 1$	85	86
16	1	21	22
	$2^{16} - 1$	52	53
8	1	21	22
	$2^8 - 1$	28	29

each stream with itself. The remainder of the matrix is symmetrical around this diagonal line, accounting for each pair of streams being considered twice (e.g.,  $(s_i, s_j)$  and  $(s_j, s_i)$ ), resulting in identical correlation coefficients  $(c_{ij} = c_{ji})$ . Consequently, only one half of the matrix (either the upper or lower triangle) contains meaningful results, while the other half comprises duplicates or '1' values.

The COEFF Engine computes only unique results using matrix parsing techniques to achieve an II of 1. We parse the indeces of  $G_p$  coefficients as if parsing the upper triangle of the square matrix as pointed by the arrows in Figure 2b following the index values from 0 to 5. Each index value is associated to two sub-indices i and *j*, where *i* represents stream  $s_i$  and acts as "parsing each row of the matrix", and j represents stream  $s_j$  and acts as "parsing each column of the matrix". For each circle  $(c_{ij})$  in Figure 2b, we retrieve from the temporally internal storage the sum of elements  $(S_{e_i}, S_{e_j})$ and sum of squares  $(S_{sq_i}, S_{sq_j})$  associated with each stream, and the sum of products associated with their pair  $S_{p_{ii}}$  (located at an address given by the index value) in order to compute the correlation coefficient. This operation is sequential and employes floating-point arithmetic to generate the correlation coefficients, which leads to a long pipeline. COEFF Engine latency lower bound  $(L_{COEFF})$  is 120 clock cycles. The difference in latency between the two engines arises from their design particularities. The ACC Engine design consists of parallel pipelines for low latency results, whereas the COEFF Engine design features a single long pipeline, resulting in higher latency as data traverses the entire pipeline.

#### 3.4 Implementation

AMNES has been implemented in C++ as a Vitis HLS (v2022.1) compute kernel and deployed on the FPGA together with Coyote [43], an open source FPGA shell. Coyote establishes streaming interfaces between both the DMA (Direct Memory Access)/Bridge Subsystem for PCI Express® [78] or an RDMA network stack and the compute kernel and implements the virtual memory management and the synchronization with the host CPU. Besides the AXI4-Stream interfaces (hls::stream<ap\_axiu<512,0,0,0>>) to stream data into and out of the kernel, the compute kernel also exposes an AXI4-Lite register interface (s\_axilite) that allows software to access kernel



(a) Correlation coefficient matrix. (b) COEFF Engine - Index parsing. Figure 2: Correlation matrix vs. COEFF Engine for 4 streams.



Figure 3: AMNES implementation.

configuration and parameter data. We use these registers to program the number of items and streams correlation is computed on. Both the AMNES kernel and the Coyote shell offer versatility across AMD Alveo's portfolio of data center accelerator cards (i.e., U50, U55C, U200, U250, U280). We have deployed AMNES together with Coyote on three of these cards, U250, U280 and U55C, and achieved a maximum operating frequency of 300 MHz.

The flow of the implementation is illustrated in Figure 3. Even if the cacheline accommodates from 16 to 64 streams for our engine, we illustrate the implementation for only 4 streams  $(s_0, s_1, s_2, s_3)$  for simplicity. However, the insights apply to the number of streams that fit into a cacheline for any data representation. When a new cacheline arrives, the pre-processing stage augments it with an asserted 'last' signal if the cacheline contains the last elements of the streams, otherwise the signal remains deasserted. This stage is combinatorial and takes only one clock cycle. The function call to the ACC Engine encapsulates through a lamba expression the partitioning of the cacheline into individual stream values. These values are used to simultaneously update the mesh of accumulators and multiply-accumulate units of the ACC Engine (Figure 3). The 3stage multiply-accumulate pipeline depth is projected onto the plain accumulators (1 stage) to match operational latencies. The ACC Engine's accumulators' state is kept in the FPGA's fabric registers.

When an asserted 'last' signal is encountered, all the obtained values in the ACC Engine are read and temporally moved to the internal memory of the FPGA (BRAM). This temporary storage enables the instant reset of all accumulators to '0' such that new incoming data can be processed immediately. The COEFF Engine reads the BRAM addresses indicated by the row, column for  $S_e$ ,  $S_{sq}$ , respectively, and the memory location indexed by  $S_p$ . For each *index* <  $G_p$ , five values are fetched and used to compute the floating-point PCC associated to the index. The COEFF Engine leverages the power of HLS to implement Equation 2 from a regular high-level floating-point expression. Our method of index parsing ensures an II = 1, guaranteeing no back pressure from the COEFF Engine on preceding modules. Correlation coefficients are collected from the engine and sent to the host CPU in the post-processing stage.

#### **4 EVALUATION**

## 4.1 Experimental Setting

We have evaluated AMNES on the AMD Heterogeneous Accelerated Compute Cluster (HACC) at ETH Zurich using the U55C data center accelerator card with the FPGA in two configurations: (1) as a coprocessor - data to be correlated is produced by the host CPU and resides in its memory, and (2) as a SmartNIC - data is produced



Figure 4: AMNES evaluation setup.

by a remote CPU and moved via RDMA-Write operations. The two configurations are represented in Figure 4 and comprise the FPGA and host CPU as compute units. The host CPU is also called the local CPU. In both cases, the FPGA is connected to the CPU via a PCIe Gen3x16 link. The SmartNIC configuration differs from the coprocessor one by enabling the RDMA stack in the Coyote framework on the FPGA. For each of the configurations, we compare the FPGA performance with the local CPU baseline performance in terms of throughput and latency. Irrespective of the configuration, the local CPU baseline reads the streams' values from its DDR memory; with the way the DDR memory gets populated being different for each configuration: (1) the CPU populates the DDR memory before executing the correlation computation, and (2) the DDR memory is populated by a remote CPU via RDMA-Write transfers.

#### 4.2 Baselines

We have developed two multi-threaded software baselines for the backend computation. One baseline collects the sufficient statistics in the same way as the FPGA implementation, and another extracts them from a matrix-matrix multiplication operation. If the first one is a standalone implementation, the second uses the Eigen Library (v3.4.0) [25], a C++ template library highly optimized for linear algebra, specifically for CPU-based matrix-matrix multiplication operations. We will henceforth refer to them as noEigen and Eigen, respectively. The Eigen Library offers explicit vectorization, compiler support (C++14), a straightforward integration with C++ code, adaptable matrix sizes and numeric types. It expects the stream values to be stored in column-major layout to optimize data partitioning between the processing threads. Irrespective of the baseline, each thread receives a data chunk size inversely proportional to the number of threads used for computation. For the noEigen baseline, the values are distributed to the corresponding accumulators, whereas for the Eigen baseline, each thread maps the received values to a matrix whose number of rows is the number of streams plus one (M+1) and the number of columns corresponds to the length of the stream segment associated with the thread, i.e. N/thread\_count. Each thread's matrix has an additional padded row of '1's to compute the sum of elements (Se) for each stream. Figure 5 illustrates the matrix-matrix multiplication between the padded streams matrix A and its transpose B when one thread is allocated for 4 streams  $(s_0, s_1, s_2, s_3)$  with N items. The transposed matrix is obtained using Eigen's .transpose() function. Still, the transpose is not materialized in memory, but returns a proxy object without doing the actual transposition. For a single thread, the sufficient statistics are directly extracted from the matrix product (i.e., the resulting matrix). For multiple threads, the matrix products

1	1	•••	1	$[s_{0N-1}]$	$s_{1_{N-1}}$	$s_{2N-1}$ B	$s_{3N-1}$	1]	$S_{e_0}$	$S_{e_1}$	$S_{e_2}$	$S_{e_3}$	N
s <sub>30</sub>	$s_{3_1}$		$s_{3_{N-1}}$	:	:	:	:	:	$S_{p_{30}}$	$S_{p_{31}}$	$S_{p_{32}}$	$S_{sq_3}$	$S_{e_3}$
s20	$s_{2_1}$		$s_{2N-1}$	× .					$=  S_{p_{20}} $	Sp21	Ssa2	Spm	Se,
$s_{1_0}$	$s_{1_1}$	• • •	$s_{1_{N-1}}$	So.	S1.	\$2.	S3.	1	$S_{p_{10}}$	$S_{sq_1}$	$S_{p_{12}}$	$S_{p_{13}}$	Se1
[s00	$s_{0_1}$	• • •	s0 <sub>N-1</sub>	[ so.	S1.	\$2.	S3.	1]	$S_{sq_0}$	$S_{p_{01}}$	$S_{p_{02}}$	$S_{p_{03}}$	$S_{e_0}$

Figure 5: Eigen matrix-matrix multiplication operation between the padded streams matrix (A) and its transpose (B).

of each thread undergo a subsequent summation process into a single (M + 1) square matrix to obtain the sufficient statistics. Then an upper module extracts them and computes the coefficients. We deploy the multi-threaded C++ implementation on the AMD EPYC 7302P 16-Core Processor@ 3.0 GHz base frequency, with each two adjacent cores sharing 16 MB of L3 Cache, and each core having 512 kB L2 Cache, and 32 kB of data and instruction L1 Cache.

#### 4.3 Datasets

For evaluating AMNES, we use synthetic datasets with arbitrary precision (ap) integer data types representations: ap\_uint\_8, ap\_uint\_16 and ap\_uint\_32 [79], and vary the streams' length from 1000 to 2 million items per stream. The parallel stream analysis depends on the number of streams fitting into a cacheline (512 bits) for a given data type (64 streams for ap uint 8, 32 streams for ap uint 16, and 16 streams for ap\_uint\_32). We opt for unsigned integer data representations (ap uint x) as they stretch the FPGA resource utilization the most out of the fixed-size data representations: fixed-point decimal (ap\_[u]fixed) and signed (ap\_int). The data volume across PCIe or RDMA networks stays constant (i.e., the bytes count) regardless of data width representation. Even if we evaluate all the pipelines with uniform data layout (same width and type), a practical choice for vector DBs and ML systems, AMNES is not restricted to this. The FPGA can be programmed to associate a different data type (integer or fixed-point decimal) and width to each pipeline (stream), e.g., for a 512-bit cacheline, 4 streams could be associated to 32-bit signed integer, 4 streams to 32-bit fixed-point decimal and 16 other streams to be associated to 16-bit unsigned. However, if floating-point values are to be employed, they have to be converted to fixed-point representation beforehand, using a dedicated AMD-Xilinx IP [5].

The evaluation setup for the two use cases (coprocessor, Smart-NIC) is illustrated in Figure 4, with (1) marking the source of the data values when entering our heterogeneous compute node composed of the host CPU and the FPGA. AMNES's working frequency depends on the chosen data width representation as it is stated in column AMNES Operating Frequency [MHz] (Fop) in Table 2. The degradation of the operational frequency is due to two factors. On the one hand, the number of MAC units required for the sum of products grows quadratically with the number of analyzed streams. On the other hand, the spatial architecture of the FPGA has to ensure all the wiring that is required between the accumulators, multiply accumulators and the rest of the system. Given the working frequency and the fact that AMNES works at cacheline granularity, we can compute the theoretical upper bound of the throughput  $(Th_{theoretical[GB/s]} = F_{op} * 64B/1000)$ . More specifically, 19.20 GB/s for AMNES's 32-bit data width and 300 MHz Fop, 16.00 GB/s for AMNES's 16-bit data width and 250 MHz Fop, and finally, 12.16 GB/s for AMNES's 8-bit data width and 190 MHz  $F_{op}$ .



Figure 6: Compute time (log scale) of PCC for 16 streams on various platforms (32-bit integer data).

The normal data distribution condition is intrinsic to the PCC algorithm to guarantee a reliable result. AMNES's efficiency is not impacted by the distribution as the system processes input data every clock cycle and updates the underlying accumulators network.

CPUs and GPUs are software-programmable fixed architectures. With the emerge of customized data types and reduced resolution representations, their inherent architectural advantage becomes debatable [31]. Data center vendors (Microsoft, Amazon, Baidu) are now focusing on FPGA's programmable data-width capability for ML deployments in the detriment of fixed architectural paths [23].

In Figure 6, we depict the compute time required on 3 individual platforms (FPGA, CPU and GPU) to obtain the PCC for 16 streams while varying each stream's length from 1 Ki to 2 Mi 32-bit integer items. For the GPU, the application utilizes the PyTorch library (torch.corrcoef [61]) on a TITAN RTX Nvidia GPU, whereas for the CPU, we utilize the two baselines and allocate only one thread for compute. Although the FPGA operates at a lower clock frequency (MHz range) compared to the CPU and GPU, its spatial pipeline customization allows for parallel processing of multiple items, providing it with an advantage. On the other hand, the GPU's clear advantage is hindered by the data movement costs.



(a) PCC for 2, 4, 8 and 16 streams (2 million 32-bit data items per each attribute (stream).



(b) PCC for 2 streams and 32-bit data representation with Postgres.

Figure 7: PCC compute time (log scale) using Postgres, Snowflake, and MySQL.

#### 4.4 Comparison with Relational Operators

To provide a performance reference for relational engines supporting PCC computation, we assess 3 engines: Postgres, Snowflake, and MySQL. Postgres and Snowflake offer a *corr(X,Y)* aggregate operator (X and Y-the two attributes), whereas MySQL queries for all Equation 2 items in the SELECT statement. Since DBMSs offer 2-by-2 correlations, we examine their efficiency from 2 up to 16 attributes for 2 million 32-bit integers in Figure 7a. For Snowflake, we use an X-Small deployment [68] and report only the execution time out of the total time (compilation + execution), which is significantly smaller than the compilation time ( $2 \times$  to  $30 \times$ ). Irrespective of the DBMS, the PCC compute time for two attributes is an order of magnitude larger than AMNES's for 2 million items and 16 streams (if we keep the system as-is, computing only 2 attributes, leaving the other 14 unused). The FPGA compute time increase in Figure 7b is due to the unused slots. The MySQL query complexity does not compete with either the other two DBs or the FPGA. Since Postgres has the fastest compute time for 2 attributes, we analyze its behavior for down to 1Ki items per attribute (Figure 7b) and observe that the compute time remains  $5 \times$  larger than for AMNES.

## 4.5 Correlation on a Coprocessor

In Figure 4a, the values that are streamed from the host's DDR to AMNES ( $(2 \rightarrow 3)$ ) via DMA-Write transfers are augmented with a 'last' signal before AMNES's compute engine entry. The 'last' signal serves as a control signal accompanying the data in the compute engine, indicating the last value of each stream. Once the FPGA

computes the PCC values, they are streamed back to the host's DDR ((3)->(4)) via DMA-Read transfers from where they can be further used ((5)). The performance measurements for the FPGA-coprocessor include the (2)->(3)->(4)->(5) data links and are illustrated in Figure 8, where each sub-figure corresponds to a different data width and parallel stream analysis for cacheline granularity: 32-bits (Figure 8a), 16-bits (Figure 8b) and 8-bits (Figure 8c), respectively. For each sub-figure, the measurements encompass the clock cycles for PCIe data transfer, as well as for collecting the sufficient statistics and computing the PCC values for each pair of streams.

In Figure 8, the throughput saturates for 8-bit data around 12 GB/s, near AMNES's theoretical upper bound of 12.16 GB/s for this data. For 16-bit and 32-bit data, the throughput saturates at values around 12.5 GB/s, lower than AMNES's theoretical upper bounds of 16 GB/s and 19.2 GB/s, respectively, but matching the bandwidth exposed by the Coyote framework to the compute kernel. Coyote's setup cost dominates the measurement for small stream lengths and becomes negligible for streams with more than 32K items, as it is illustrated by the throughput graphs in Figure 8. The lower working frequency of the compute kernel impacts latency measurements for 1Ki items per stream. Among the three data representations, the compute kernel operating at 300 MHz exhibits the shortest compute time (Figure 8a), while the one operating at 190 MHz has the longest time for the same data volume (Figure 8c).

Regardless of the data width representation, the FPGA implementation outperforms the multi-threaded C++ baselines (Eigen or noEigen). The AMNES-like collection baseline (noEigen) performs



Figure 8: Compute time (bar, left y-axis, log scale) and throughput (line, right y-axis, linear scale) for RAM-RAM experiments for 16/32/64 streams analyzed in parallel on the FPGA; including *collecting* the sufficient statistics and computing *pcc* values.



Figure 9: Compute time (bars, left y-axis, log scale) and throughput (line, right y-axis, linear scale) measurements for RAM-RAM experiments with 16/32/64 parallel streams analyzed on the CPU (2 million items per stream). We differentiate the compute time measurements between sufficient statistics collection by matrix multiplication (Eigen) and AMNES-like collection (noEigen). Compute time includes collecting, merging partial values from each thread, and computing the PCC.

better than the matrix collection one (Eigen) for single and dual thread deployments, but the Eigen baseline has a better overall performance due to dense matrix-matrix product optimizations, saturating at around 6 GB/s, 5 GB/s and 2.5 GB/s (Figure 9) for 16, 32 and 64 streams and 2 million items per stream, respectively. For each baseline, we report the throughput (Figure 9-lines) and compute time (Figure 9-bar) measurements. Compute time is primarily consumed by collecting the sufficient statistics, whereas the merge and PCC calculation times are much smaller. If the former gets damped with larger number of threads allocated for the task, saturating for 16 threads and more, the latter two have a similar behavior across all data width representations.

#### 4.6 Correlation on a SmartNIC

For this use case, we utilize the setup in Figure 10a with two CPUs (remote host A initiates the RDMA-Write transfers while local host B receives them) and two data center class FPGAs connected via a switch. The data values coming from the remote host are forwarded by the RDMA network stack simultaneously to the local CPU's DDR(2) and to AMNES (2) on the FPGA as illustrated by the links (1)->(2) and (1)->(2) in Figure 4b. When the coefficients results are ready, DMA-Read transfers transfer them from AMNES to the local CPU's DDR  $((3) \rightarrow (4))$ . For the software baseline associated with the SmartNIC setup, the Eigen-based application starts processing the values as soon as they are placed in the local CPU's memory (2)->(5)), with each thread having associated to it one or multiple slices of the data. The application starts processing before full stream data transfer and once the first data values reach the CPU's DDR. The FPGA performance assessment includes the (1) - >(2) - >(3) - >(4) data links, whereas the CPU baseline for the SmartNIC setup includes (1->2)->5 data link together with the CPU processing.

We focus on the 32-bit data representation and evaluate the network setup by employing 32 KiB RDMA-Write transfers. To achieve streams length from 4Ki, 16Ki up to 2Mi for 16 streams, multiple RDMA transfers are needed. Considering the RDMA connection in Figure 10a, we distinguish between different measurable durations: (1) **network time** - the time from when the first write request is sent from the remote host till the last acknowledgment is received by it from the local CPU; (2) **data reception time** - the time from when the first data byte arrives to the local CPU's memory till the acknowledgment is sent to the remote CPU; (3) **data reception + correlation on CPU (x threads)** - the time from when the first



(b) Latency (bars, left y-axis, log scale) and throughput (line, right y-axis, linear scale).

Figure 10: RDMA communication setup and performance.

byte arrives to the local CPU's memory till the correlation coefficients are produced (1 to 16 threads are allocated to compute the sufficient statistics using Eigen Library (v3.4.0) and a single thread is allocated for the PCC values computation); (4) **data reception + correlation on FPGA** - the time from when the first byte arrives to the local CPU's memory, until all the coefficients are transferred from the FPGA to the local CPU's memory via PCIe transfers.

In Figure 10b, we assess network throughput by adjusting the number of transfers exchanged between the remote and local CPUs. The number of transfers is varied from 8 for 4Ki items per stream, 2048 for 1Mi items per stream, to 4096 for 2Mi items per stream. This serves as a reference point for evaluating network communication overhead. The network throughput saturates when the number of transfers exchanged between the two CPUs exceeds 64. At this point, the throughput stabilizes at approximately 12.5 GB/s, which aligns with the PCIe saturation. This confirms that the Coyote framework does not impose any backpressure on the network. We present the latency measurements as the data reception time instead of the network time. The data reception time includes the network time for all transfers except the first one. For 16 streams, the data reception time ranges from 21 µs for 4Ki items per stream to 11 ms for 2Mi items per stream. Both AMNES and the CPU baseline start data processing as soon as the data enters the system, in



Figure 11: RDMA communication w/o correlation computation time on the CPU and FPGA (bars, left y-axis, log scale).

the FPGA and CPU's DDR, respectively, rendering the first network transfer time insignificant. Modifications to the Eigen-baseline allow processing of smaller data chunks than N/thread\_count (i.e., 16 items/stream) without waiting for the entire transfer.

On the FPGA, analyzing 16 streams produces 120 correlation coefficient floating-point values. This requires 8 DMA-Read transfers and only one PCIe transfer for a 4096 B payload size. This communication expense, combined with RDMA, incurs an overhead of approximately 1  $\mu$ s. As shown in Figure 11 (rdma and rdma+corr FPGA bars), this overhead results in latency similar to solely receiving the data. In Figure 11, we analyze stream lengths up to 1Mi items per stream, which already saturate the available bandwidth.

Implementing correlation on the host CPU incurs significantly higher time overhead compared to simply receiving the data (baseline). On average, it is  $30 \times$  larger when a single thread (*rdma+corr* CPU: 1 thread bar in Figure 11) is allocated for getting the sufficient statistics and computing the coefficients. Allocating more compute threads reduces this overhead. For 8 threads, it is only 7× larger on average than the baseline (rdma+corr CPU: 8 threads bar in Figure 11). The compute time on the host CPU decreases on average by 1.62× when the number of threads allocated for the task is doubled. However, even with 16 threads, the host CPU implementation is on average 4.4× slower than the FPGA implementation (i.e., AMNES). The FPGA incurs no backpressure on the network by processing data at every clock cycle (II = 1). Even if the host CPU implementation is slower than the FPGA implementation, it does not have a negative impact on the RDMA network performance since the CPU acts upon data already residing in its DDR memory and transferred there via RDMA-Write transactions that do not involve the CPU.

For a 32-bit data representation, the latency inferred by AMNES in the SmartNIC setup is larger than the latency inferred in the coprocessor setup with a  $\Delta < 100 \,\mu s$ . This difference ( $\Delta$ ) becomes larger as the number of items per streams increases, since more RDMA-Write transfers are needed to transfer all the data.

## **5 DISCUSSION**

In this section, we discuss the number of streams used by other systems and explain how AMNES can be generalized to more than 16/32/64 streams, why C++ HLS was the language of choice, the challenges the FPGA must overcome, and how the use of the gathered sufficient statistics can be further extended.

System	Parallel Attributes	Total Number of Attributes	Items per attribute	РСС
AMNES	16-64	16-64	2 Mi	🗸
BRAID [66]	2	59	100 Ki	🗸
Joglekar et al. [35]	2	5	45 Ki	-
Hermit [77]	2	16 200	4 Mi 15 Ki	-
CORDS [30]	-	18	1 Ki - 64 Ki - 2 Mi	-
Wadjet [65]	-	5000	50 Ki	🗸
EXORD <sup>+</sup> [50]	-	8	90 Ki	-
COCOA [20]	2	22	100 - 356 Ki	ranŀ
Joglekar et al. [35]	-	5	45 Ki	-

Table 4: Number of attributes (streams).

#### 5.1 Number of Streams

In our evaluation, we analyze AMNES's performance for a maximum of 64 streams in parallel. For reference, in Table 4 we summarize the parameters used in related work, covering the number of coefficients computed in parallel, the total number of streams/attributes considered, and the size of each stream. AMNES is the only one computing coefficients in parallel (16 to 64 while all others are just between pairs of attributes). Similarly, our experiments consider much larger data sets per stream/attribute than almost in all previous work, especially when compared to those computing PCC instead of simpler or approximated forms of correlation.

## 5.2 Engine Generalization

Engine generalization expands AMNES's capabilities to more than 16/32/64 streams by trading latency, resources, or complexity and is defined by the extended parameters in Table 5. As before, each cacheline (512 bits) is composed of values from M streams. Apart from these definitions, we introduce the notion of batch of streams. A batch of streams (a *batch*), represents the number of cachelines needed to cover one value from each distinct stream M. If the total number of streams to analyze exceeds the cacheline capacity (S > M), then a batch extends over  $\lceil S/M \rceil$  cachelines. We focus on generalizing the ACC engine due to its parallelizability and low latency. Once all the sufficient statistics are gathered, they can be sent to the host CPU and queried for the PCC value. We have identified two approaches to generalize the concept: (1) support a pre-defined total number of streams that are time-multiplexed at the cacheline level (timeAMNES ACC), or (2) impose fixed hardware limitations of the engine and run stream values multiple times through it (fixedAMNES ACC); the latter solution requires all streams' values to be temporally stored in internal or external FPGA memory.

timeAMNES ACC-Sufficient Statistics. Figure 12 shows the principle for this generalization. timeAMNES ACC differs from the basic engine by supporting (M+1) streams instead of M. The extra one stream is a dummy stream (d) at the beginning, when the batch's first cacheline arrives (Figure 12a). With the arrival of subsequent batch cachelines, the dummy stream's place is taken by the values of each previously seen stream (Figure 12b). The procedure is as follows: (1) one cacheline arrives and goes through the ACC engine; (2) ACC computes the required sufficient statistics for (M+1) streams, and discards the sums that are part of the dummy component or have already been computed; (3) update BRAM storage for sum of elements ( $S_e$ ), sum of squares ( $S_{sq}$ ) and sum of products  $(S_p)$ ; (4) update line storage for the later use of line's values; (5) replace the dummy stream with the first element of the line storage, and send it together with the new incoming cacheline through the ACC engine (Figure 12b); (6) repeat this process until all the components of the previous line storage have been

Table 5: Symbols for design generalization, extends Table 1.

S	Total number of streams ( $S \ge 4$ ).
$T_{CLK}$	Clock period of the design.
$T_p$	Number of distinct pairs between S streams.
$C_{S_t}$	Cumulative streams observed at time $t$ ( $C_{S_0} = M$ , $t = 0$ ).
State <sub>ACC</sub>	Accumulator state to maintain.



Figure 12: timeAMNES ACC.

combined with the new incoming cacheline. The line storage grows in a cumulative way, with cacheline elements being added to it until all streams' values in a batch have been seen. When a new batch of streams arrives, the line storage and the dummy stream are considered empty. We distinguish two actions for a data cacheline observed at a moment t: (1) updates - immediate action (Equation 3), and (2) holds - future action (Equation 4).

$$updates = \begin{cases} \frac{M(M+3)}{2} , t = 0\\ \frac{M(M+3)}{2} + M * C_{S_t}, t \ge 1 \end{cases}$$
(3)

$$holds = \frac{S(S-1)}{2} - \frac{C_{S_t}(C_{S_t}-1)}{2} - \frac{(S-C_{S_t})(S-C_{S_t}-1)}{2}$$
(4)

For each stream *i* in a cacheline, the updates include:  $S_{e_i}$ ,  $S_{sq_i}$ ,  $S_{p_i}$  for the distinct pairs that can be formed between the streams of the cacheline, and  $S_{p_i}$  for the pairs that have been on hold and have the second stream (operator) within the given cacheline. The holds are a metric for future actions and include all pairwise sum of products that can be formed between current cacheline streams and batch streams, but are missing the second stream (operator).

Depending on the total number of streams, the accumulator state  $State_{ACC}$  can be maintained on-chip or off-chip. The number of accumulators involved in the computation has a quadratic relationship with the total number of streams.  $State_{ACC} = \frac{S(S+3)}{2} * Width_{ACC}$ .

We extrapolate in Table 6 how many streams would fit in the internal memory (BRAM or/and URAM) for 64-bit accumulators for AMD data center accelerator cards. We also consider the deployment of the engine in a single FPGA SLR (Super Logic Region-a single FPGA die slice) instead of the entire FPGA in the last table's column. Each data center class FPGA has between 2 to 4 SLRs.

TimeAMNES<sub>ACC</sub>'s latency (Equation 5) indicates the back pressure that will be induced for large *S* values, making this solution inappropriate for high rate communication links.

$$Total_{latency} = L_{ACC} + \left[\frac{S(S-M)}{2M} + 1\right] * T_{CLK}$$
(5)

**fixedAMNES** <sub>ACC</sub>-Sufficient Statistics. As opposed to the previous solution that time-multiplexes stream values, fixedAMNES <sub>ACC</sub> analyzes all the values from the streams covered in one cacheline

Table 6: AMD data center accelerator cards.

Board	BRAM [MB]	URAM [MB]	<b>S</b> <sub>BRAM</sub>	<b>S</b> <sub>URAM</sub>	S <sub>SLR:BRAM+URAM</sub>
U50	6.048	23.04	1228	2398	1754
U200	9.72	34.56	1557	2937	1912
U250	12.096	46.08	1737	3392	1853
U280	9.072	34.56	1504	2937	1856
U55c	8.8625	33.75	1486	2903	1883

before moving to another set of streams. This solution constrains the engine's compute and storage capabilities at M streams. If we have  $T_p = S(S-1)/2$ , the total number of distinct pairs between S streams, and  $G_p = M(M-1)/2$ , the number of distinct pairs between M streams, then the lower bound for the number of passes through the engine to obtain all necessary pairs would be  $T_p/G_p = [S(S-1)]/[M(M-1)]$ . This lower bound is not a tight bound, and heuristics indicate more passes for full pair coverage [3]. For example, for S = 6 and M = 3, 6 passes instead of 5 are needed in order to cover all the streams. This solution requires the stream values to be stored, so that the set of streams of one cacheline can be correlated with the set of streams of another cacheline, after all values of the previous set have been seen by the engine. This offers an upper bound for the supported number of streams that can be stored in the FPGA external memory: for DRAM, a maximum of 2000 streams of 1 million items each or 250 streams of 8 million items each; for HBM, a maximum of 16000 streams of 1 million items each or 2000 streams of 8 million items each, for 32-bit data value representations. The latency of the pairing compute will depend on the number of groups that need to be formed in order to create all the pairs that are necessary. The advantages of this solution are that it will not add back pressure on the communication link and will require only the state of one engine to be maintained inside the FPGA. The solution's disadvantages will be: (1) all streams values need to be temporally stored in the memory and passed through the engine; (2) a data structure that tracks formed pairs, potentially solvable through edge clique covering [3, 15]. If we consider the streams being the vertices of a graph and the pair between each two streams being the edges, then all the unique pairs between S streams are characterized by a fully connected graph with S vertices. If the same analogy is applied to ACC Engine capabilities, creating a fully connected graph with M vertices, then the engine becomes a clique of the total number of supported streams.

## 5.3 Challenges of Using FPGAs

Utilizing FPGAs for accelerating database tasks requires addressing challenges that are unique to FPGAs but absent in CPUs. FPGAs implement computations as a spatial dataflow architecture, where the compute operators are connected to physically resemble the dataflow graph using actual point-to-point links rather than communicating through shared address-based memory resources like in a CPU. Such a spatial control flow enables greater degrees of parallelism but also requires to design the algorithm accordingly.

Deploying a design on an FPGA involves synthesis tools for *place and route*: positioning of operators on the FPGA fabric and the links between them. The implemented routing defines the signal propagation latencies that limit the clock frequency achievable by the design. AMNES's operational frequency is 300 MHz, low compared with CPUs but competitive for FPGAs. For performance, the design needs to exploit the hardware customization and fully tailored parallelism possible on an FPGA. In our case, through the processing of multiple input streams in parallel.

AMNES is implemented in C++ using Vitis high-level synthesis (HLS) capabilities. HLS code is more approachable and manageable than, e.g., VHDL or (System)Verilog but it still requires awareness of the underlying spatial architecture. Pragmas are widely employed to guide the synthesis with hints on loop unrolling, pipelining, dataflow regioning and resource mapping. The goal is to ingest one input vector in each and every clock cycle, i.e., to sustain an initiation interval (II) of 1. Our design can serve as a blueprint for other researchers working with FPGAs.

The biggest advantage of having correlation computation on the FPGA is that data can be processed on the fly, as it arrives from the network, without CPU involvement. Not all algorithms and database operators can do this, so FPGAs are not suitable for all operations a database engines can do. The results of the paper show, however, the advantages of an FPGA based accelerator for operations that can be streamed such as the correlation. The limitation associated to the design is the amount of streams that can be processed in parallel before the design is not able to maintain 100 Gbps for PCIe and/or network. AMNES is capable of correlating 64 parallel streams of 8-bit data with an initiation interval of 1 at 190 MHz, enough to match the net data throughput just below PCIe/network capabilities (Figure 8c).

Regarding integration in a full system, our approach aligns with a recent proposal using the FPGA on the I/O path, as an external service for compression and encryption [13]. In our use case, data arrives from the network.

#### 5.4 Data Representation

AMNES is adaptable to different data representations and is not tied to the ones that are used in this paper for illustration purposes. Adapting to different representations may require temporary buffering and simple transformations. For instance, systems such as Snowflake employ a format where groups of rows are mapped into individual micro-partitions, organized in a columnar fashion [69]. Applying our design would require to transpose the micro-partition to reconstruct the tuple. This can be easily done on the fly on an FPGA without any performance loss. This is the reverse (columns to rows) of the transformation performed by Oracle Exadata (rows to columns). In the storage layer of Oracle Exadata, AMNES would not need to transpose the data. In addition to columns-to-rows transformations, FPGAs can efficiently decompress (e.g., delta encoding or run-length) and decrypt column-based structures [36]. Such operations can be easily integrated with our design.

For varying data arrival rates from different streams, the cachelines are assembled in a temporary input buffer. Faster source streams will eventually experience pacing backpressure in this process. For data types smaller than 8 bits (i.e., more than 64 items in a cacheline), that exceed AMNES's compute capacity, the input data needs to be divided to match the engine size. Then multiple iterations over the data are required so that all streams are mutually correlated. This introduces latency but can be addressed as the next case when data is wider than a cacheline.

Data spread across multiple cachelines can currently be handled by instantiating multiple AMNES engines within the FPGA, with one or two engines being allocated to each FPGA SLR and each engine processing one cacheline at a time. For example, for 32-bit data representation, a maximum of 6 engines can be instantiated on the FPGA, with each engine occupying only half an SLR (the Alveo U55c board comprises 3 SLRs). This implies processing data spread across 6 cachelines, where each engine correlates 16 streams at a time. Once the data from the 6 cachelines is analyzed ( $6 \times 120 = 720$  PCCs) and placed in the CPU memory (RDMA use case), the CPU application can trigger the 6 engines to compute the correlation coefficient by combining streams from different cachelines.

#### 5.5 Further Usage

The Kendall and Spearman correlation coefficients are nonparametric statistical tests that rely on the data ranks rather than the values themselves, requiring data to be sorted beforehand. The Spearman coefficient  $(r_S)$  can be calculated using AMNES when the values are rank values associated with the stream values. Nevertheless, if all ranks are distinct integers, the Spearman coefficient is reduced to  $r_S = 1 - \frac{6\sum_{t=0}^{N-1} d_t^2}{N(N^2 - 1)}$ , where  $d_t$  represents the difference between the ranks of two observations. This formulation is easier to implement on FPGA since it requires fewer accumulators and multipliers than AMNES. The same ranked data can be applied to the computation of the Kendall coefficient ( $\tau$ ), which then reduces to comparisons and unitary additions, namely to  $\frac{n_c - n_d}{\frac{1}{2}N(N-1)}$ , where  $n_c$  represents the number of observations  $\tau =$ ordered in the same way, and  $n_d$  represents the number of observations ordered differently. The sufficient statistics gathered in the ACC Engine can be used to estimate a simple linear regression, a statistical function commonly available in relational engines (e.g., Oracle's REGR SLOPE operator). If we consider the general line equation Y = mX + n, knowing the stream values Benchar me equation t = mX + n, knowing the stream values  $X\left(\sum_{t=0}^{N-1} x_t\right)$  and  $Y\left(\sum_{t=0}^{N-1} y_t\right)$ , then  $n = \frac{\sum_{t=0}^{N-1} y_t}{N} - m * \frac{\sum_{t=0}^{N-1} x_t}{N}$ , and  $m = \frac{N\sum_{t=0}^{N-1} x_t y_t - \sum_{t=0}^{N-1} x_t \sum_{t=0}^{N-1} y_t}{N\sum_{t=0}^{N-1} x_t^2 - (\sum_{t=0}^{N-1} x_t)^2}$  by using the least squares method for a set of paired data [29]. For only Y = mX, m is calculated with  $\sum_{t=0}^{N-1} x_t y_t$ .  $m = \frac{\sum_{t=0}^{N-1} x_t y_t}{\sum_{t=0}^{N-1} x_t^2}$ . The ACC Engine, notably its sum of products and sum of squares, can be used for cosine similarity (CosSim) computation [57], since  $CosSim(X, Y) = \frac{\sum_{t=0}^{N-1} x_t y_t}{\sqrt{\sum_{t=0}^{N-1} x_t^2} \sqrt{\sum_{t=0}^{N-1} y_t^2}}$  for two

streams X and Y. In turn, this can be used to derive the cosine distance (1-CosSim) between vectors in a multi-dimensional space [57]. Similarly, the statistics computed can be trivially employed to derive other valuable meta-data such as variance, standard deviation, or averages. With very little overhead, functions such as min, max, top/bottom n outliers can be easily added to the overall design.

#### **6** CONCLUSION

This paper explores the implementation space of advanced data analytics while leveraging specialized hardware solutions. We focus on correlation and show its integration as a coprocessor or on a SmartNIC with an RDMA network interface, all without impacting CPU or network performance. Maximum advantages emerge when correlation is deployed within the network, as additional and unnecessary data transfers via PCIe are avoided. In this scenario, FPGA outperforms multi-threaded CPU execution by 4.4× on average.

## ACKNOWLEDGMENTS

We would like to thank AMD for the generous donation of the Heterogeneous Accelerated Compute Clusters (HACCs) at ETH Zurich and the financial support for the research at the Systems Group.

#### REFERENCES

- Statology 2021. Correlation Does Not Imply Causation: 5 Real-World Examples. Statology. https://www.statology.org/correlation-does-not-imply-causationexamples/
- [2] Marcos K. Aguilera, Naama Ben-David, Rachid Guerraoui, Antoine Murat, Athanasios Xygkis, and Igor Zablotchi. 2023. uBFT: Microsecond-Scale BFT using Disaggregated Memory. In Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2023, Vol. 2. ACM, 862–877. https://doi.org/10.1145/3575693.3575732
- [3] Noga Alon, Yair Caro, and Raphael Yuster. 1998. Packing and covering dense graphs. Journal of Combinatorial Designs 6, 6 (1998), 451–472. https://doi.org/10. 1002/(SICI)1520-6610(1998)6:6<451::AID-JCD6>3.0.CO;2-E
- [4] Meteb M. Altaf, Eball H. Ahmad, Wei Li, Houxiang Zhang, Guoyuan Li, and Changshun Yuan. 2015. An ultra-high-speed FPGA based digital correlation processor. *IEICE Electron. Express* 12 (2015). https://doi.org/10.1587/elex.12.20150214
- [5] AMD-Xilinx. [n.d.]. Floating-Point Operator v7.1 Product Guide (PG060). Retrieved May 17, 2023 from https://docs.xilinx.com/v/u/en-US/pg060-floating-point
   [6] Buda Andrzej and Jarynowski Andrzej. 2010. Life-time of correlations and its
- applications.
   [7] Jeff Barr. 2021. AQUA (Advanced Query Accelerator) A Speed Boost for Your
- [7] Jeff Barr. 2021. AQUA (Advanced Query Accelerator) A Speed Boost for Your Amazon Redshift Queries. Retrieved May 10, 2023 from https://aws.amazon.com/ blogs/aws/new-aqua-advanced-query-accelerator-for-amazon-redshift/
- [8] Tim Bock. [n.d.]. What is a Correlation Matrix? Retrieved November 03, 2022 from https://www.displayr.com/what-is-a-correlation-matrix/
- [9] Sebastian Breß, Felix Beier, Hannes Rauhe, Kai-Uwe Sattler, Eike Schallehn, and Gunter Saake. 2013. Efficient co-processor utilization in database query processing. Inf. Syst. 38, 8 (2013), 1084–1096. https://doi.org/10.1016/j.is.2013.05.004
- [10] Paul Brown and Peter J. Haas. 2003. BHUNT: Automatic Discovery of Fuzzy Algebraic Constraints in Relational Data. In Proceedings of 29th International Conference on Very Large Data Bases, VLDB 2003. Morgan Kaufmann, 668–679. https://doi.org/10.1016/B978-012722442-8/50065-3
- [11] Lei Cao and Elke A. Rundensteiner. 2013. High Performance Stream Query Processing With Correlation-Aware Partitioning. Proc. VLDB Endow. 7, 4 (2013), 265–276. https://doi.org/10.14778/2732240.2732245
- [12] Dar-Jen Chang, Ahmed H. Desoky, Ming Ouyang, and Eric C. Rouchka. 2009. Compute Pairwise Manhattan Distance and Pearson Correlation Coefficient of Data Points with GPU. In 10th ACIS International Conference on Software Engineering, Artificial Intelligences, Networking and Parallel/Distributed Computing, SNPD 2009. IEEE Computer Society, 501–506. https://doi.org/10.1109/SNPD.2009.34
- [13] Monica Chiosa, Fabio Maschi, Ingo Müller, Gustavo Alonso, and Norman May. 2022. Hardware Acceleration of Compression and Encryption in SAP HANA. Proc. VLDB Endow. 15, 12 (2022), 3277–3291. https://www.vldb.org/pvldb/vol15/p3277chiosa.pdf
- [14] Stavros Christodoulakis. 1983. Estimating record selectivities. Information Systems 8, 2 (1983), 105–115. https://doi.org/10.1016/0306-4379(83)90035-2
- [15] Marek Cygan, Marcin Pilipczuk, and Michal Pilipczuk. 2016. Known Algorithms for Edge Clique Cover are Probably Optimal. *SIAM J. Comput.* 45, 1 (2016), 67–83. https://doi.org/10.1137/130947076
- [16] Ruslana Dalinina. 2017. Introduction to Correlation. Oracle AI & Data Science. Retrieved March 2, 2022 from https://blogs.oracle.com/ai-and-datascience/post/ introduction-to-correlation
- [17] Jialin Ding, Vikram Nathan, Mohammad Alizadeh, and Tim Kraska. 2020. Tsunami: A Learned Multi-dimensional Index for Correlated Data and Skewed Workloads. Proc. VLDB Endow. 14, 2 (2020), 74–86. https://doi.org/10.14778/ 3425879.3425880
- [18] Aleksandar Dragojevic, Dushyanth Narayanan, Miguel Castro, and Orion Hodson. 2014. FaRM: Fast Remote Memory. In Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2014. USENIX Association, 401–414. https://www.usenix.org/conference/nsdi14/technicalsessions/dragojevi%C4%87
- [19] Anshuman Dutt, Chi Wang, Azade Nazi, Srikanth Kandula, Vivek R. Narasayya, and Surajit Chaudhuri. 2019. Selectivity Estimation for Range Predicates using Lightweight Models. *Proc. VLDB Endow.* 12, 9 (2019), 1044–1057. https: //doi.org/10.14778/3329772.3329780
- [20] Mahdi Esmailoghli, Jorge-Arnulfo Quiané-Ruiz, and Ziawasch Abedjan. 2021. COCOA: COrrelation COefficient-Aware Data Augmentation. In Proceedings of the 24th International Conference on Extending Database Technology, EDBT 2021. 331–336. https://doi.org/10.5441/002/edbt.2021.30
- [21] Arash Fard. 2019. How to Calculate a Correlation Matrix Data Exploration for Machine Learning. Vertica. https://www.vertica.com/blog/in-database-machinelearning-2-calculate-a-correlation-matrix-a-data-exploration-post/
- [22] Daniel Firestone, Andrew Putnam, Sambrama Mundkur, Derek Chiou, Alireza Dabagh, Mike Andrewartha, Hari Angepat, Vivek Bhanu, Adrian M. Caulfield, Eric S. Chung, Harish Kumar Chandrappa, Somesh Chaturmohta, Matt Humphrey, Jack Lavier, Norman Lam, Fengfen Liu, Kalin Ovtcharov, Jitu Padhye, Gautham Popuri, Shachar Raindel, Tejas Sapre, Mark Shaw, Gabriel Silva, Madhan Sivakumar, Nisheeth Srivastava, Anshuman Verma, Qasim Zuhair, Deepak

Bansal, Doug Burger, Kushagra Vaid, David A. Maltz, and Albert G. Greenberg. 2018. Azure Accelerated Networking: SmartNICs in the Public Cloud. In 15th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2018. 51–66. https://www.usenix.org/conference/nsdi18/presentation/firestone

- [23] Karl Freund. [n.d.]. Microsoft: FPGA Wins Versus Google TPUs For AI. Retrieved May 17, 2023 from https://moorinsightsstrategy.com/microsoft-fpgawins-versus-google-tpus-for-ai/
- [24] Yixiao Gao, Qiang Li, Lingbo Tang, Yongqing Xi, Pengcheng Zhang, Wenwen Peng, Bo Li, Yaohui Wu, Shaozong Liu, Lei Yan, Fei Feng, Yan Zhuang, Fan Liu, Pan Liu, Xingkui Liu, Zhongjie Wu, Junping Wu, Zheng Cao, Chen Tian, Jinbo Wu, Jiaji Zhu, Haiyong Wang, Dennis Cai, and Jiesheng Wu. 2021. When Cloud Storage Meets RDMA. In 18th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2021. 519–533. https://www.usenix.org/conference/ nsdi21/presentation/gao
- [25] Gaël Guennebaud, Benoît Jacob, et al. 2010. Eigen v3. http://eigen.tuxfamily.org
- [26] Chuanxiong Guo, Haitao Wu, Zhong Deng, Gaurav Soni, Jianxi Ye, Jitu Padhye, and Marina Lipshteyn. 2016. RDMA over Commodity Ethernet at Scale. In Proceedings of the ACM SIGCOMM Conference 2016. ACM, 202–215. https: //doi.org/10.1145/2934872.2934908
- [27] Mark A. Hall. 2000. Correlation-based Feature Selection for Discrete and Numeric Class Machine Learning. In Proceedings of the Seventeenth International Conference on Machine Learning (ICML) 2000. 359–366.
- [28] Max Heimel and Volker Markl. 2012. A First Step Towards GPU-assisted Query Optimization. In International Workshop on Accelerating Data Management Systems Using Modern Processor and Storage Architectures - ADMS 2012. 33-44. http://www.adms-conf.org/heimel\_adms12.pdf
- [29] Cynthia Helzner. 2022. Least Squares Regression: Formula, Method, and Examples. https://study.com/academy/lesson/least-squares-regression-definitionequations-examples.html
- [30] Ihab F. Ilyas, Volker Markl, Peter J. Haas, Paul Brown, and Ashraf Aboulnaga. 2004. CORDS: Automatic Discovery of Correlations and Soft Functional Dependencies. In Proceedings of the International Conference on Management of Data, SIGMOD 2004. 647–658. https://doi.org/10.1145/1007568.1007641
- [31] Intel. 2022. Compare Benefits of CPUs, GPUs, and FPGAs for Different oneAPI Compute Workloads. Retrieved May 23, 2023 from https: //www.intel.com/content/www/us/en/developer/articles/technical/comparingcpus-gpus-and-fpgas-for-oneapi.html
- [32] Zsolt István, Louis Woods, and Gustavo Alonso. 2014. Histograms as a side effect of data movement for big data. In *International Conference on Management of Data, SIGMOD 2014.* 1567–1578. https://doi.org/10.1145/2588555.2612174
- [33] Vladimir Ivanov and Todor Stoilov. 2019. Design and Implementation of Moving Average Calculations with Hardware FPGA Device. In Advanced Computing in Industrial Mathematics: 12th Annual Meeting of the Bulgarian Section of SIAM 2017. Springer, 189–197.
- [34] Carlos M. Jarque. 2011. Jarque-Bera Test. In International Encyclopedia of Statistical Science. 701–702. https://doi.org/10.1007/978-3-642-04898-2\_319
- [35] Manas Joglekar, Hector Garcia-Molina, Aditya G. Parameswaran, and Christopher Ré. 2015. Exploiting Correlations for Expensive Predicate Evaluation. In Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data 2015. 1183–1198. https://doi.org/10.1145/2723372.2723715
- [36] Kaan Kara, Ken Eguro, Ce Zhang, and Gustavo Alonso. 2018. ColumnML: Column-Store Machine Learning with On-The-Fly Data Transformation. Proc. VLDB Endow. 12, 4 (2018), 348–361. https://doi.org/10.14778/3297753.3297756
- [37] Yiping Ke, James Cheng, and Wilfred Ng. 2007. Correlation search in graph databases. In Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining 2007. 390–399. https://doi.org/10.1145/ 1281192.1281236
- [38] Hideaki Kimura, George Huo, Alexander Rasin, Samuel Madden, and Stanley B. Zdonik. 2009. Correlation Maps: A Compressed Access Method for Exploiting Soft Functional Dependencies. *Proc. VLDB Endow.* 2, 1 (2009), 1222–1233. https://doi.org/10.14778/1687627.1687765
- [39] Hideaki Kimura, George Huo, Alexander Rasin, Samuel Madden, and Stanley B. Zdonik. 2010. CORADD: Correlation Aware Database Designer for Materialized Views and Indexes. *Proc. VLDB Endow.* 3, 1 (2010), 1103–1113. https://doi.org/10.14778/1920841.1920979
- [40] Nicolas Klodt, Lars Seifert, Arthur Zahn, Katrin Casel, Davis Issac, and Tobias Friedrich. 2021. A Color-blind 3-Approximation for Chromatic Correlation Clustering and Improved Heuristics. In *The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining 2021*. 882–891. https://doi.org/10.1145/ 3447548.3467446
- [41] Michael A. Koets and Peter W. A. Roming. 2021. Computationally Efficient Image Correlation for De-blurring with Photon-Counting Instruments. In 2021 IEEE Aerospace Conference (50100). 1–8. https://doi.org/10.1109/AERO50100.2021. 9438236
- [42] Dario Korolija, Dimitrios Koutsoukos, Kimberly Keeton, Konstantin Taranov, Dejan S. Milojicic, and Gustavo Alonso. 2022. Farview: Disaggregated Memory with Operator Off-loading for Database Engines. In 12th Conference on Innovative Data Systems Research, CIDR 2022. https://www.cidrdb.org/cidr2022/papers/p11-

korolija.pdf

- [43] Dario Korolija, Timothy Roscoe, and Gustavo Alonso. 2020. Do OS abstractions make sense on FPGAs?. In 14th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2020. 991–1010. https://www.usenix.org/conference/ osdi20/presentation/roscoe
- [44] Alexander Kumaigorodski, Clemens Lutz, and Volker Markl. 2021. Fast CSV Loading Using GPUs and RDMA for In-Memory Data Processing. In Datenbanksysteme für Business, Technologie und Web (BTW) 2021 (LNI, Vol. P-311). 19–38. https://doi.org/10.18420/btw2021-01
- [45] Lak Lakshmanan. 2019. Simplified data transformations for machine learning in BigQuery. Google Cloud. https://cloud.google.com/blog/products/dataanalytics/simplified-data-transformations-for-machine-learning-in-bigquery
- [46] Se Kwon Lee, Soujanya Ponnapalli, Sharad Singhal, Marcos K. Aguilera, Kimberly Keeton, and Vijay Chidambaram. 2022. DINOMO: An Elastic, Scalable, High-Performance Key-Value Store for Disaggregated Persistent Memory (Extended Version). CoRR abs/2209.08743 (2022). https://doi.org/10.48550/arXiv.2209.08743
- [47] Bojie Li, Zhenyuan Ruan, Wencong Xiao, Yuanwei Lu, Yongqiang Xiong, Andrew Putnam, Enhong Chen, and Lintao Zhang. 2017. KV-Direct: High-Performance In-Memory Key-Value Store with Programmable NIC. In Proceedings of the 26th Symposium on Operating Systems Principles 2017. 137–152. https://doi.org/10.1145/3132747.3132756
- [48] Yuhong Li, Leong Hou U, Man Lung Yiu, and Zhiguo Gong. 2016. Efficient discovery of longest-lasting correlation in sequence databases. VLDB Journal 25, 6 (2016), 767-790. https://doi.org/10.1007/s00778-016-0432-7
- [49] Allison Linn. 2018. Real-time AI: Microsoft announces preview of Project Brainwave. Microsoft. https://blogs.microsoft.com/ai/build-2018-project-brainwave/
- [50] Yuchen Liu, Hai Liu, Dongqing Xiao, and Mohamed Y. Eltabakh. 2018. Adaptive correlation exploitation in big data query optimization. VLDB Journal 27, 6 (2018), 873-898. https://doi.org/10.1007/s00778-018-0515-8
- [51] Guy Lohman. 2014. Is Query Optimization a "Solved" Problem? ACM SIGMOD Blog. https://wp.sigmod.org/?p=1075
- [52] Bernard Marr. 2016. Key Business Analytics: The 60+ business analysis tools every manager needs to know. Pearson UK.
- [53] Matt McGee. 2014. Microsoft's Catapult Project Aims To Speed Bing Search, Improve Relevancy. SearchEngineLand. https://searchengineland.com/microsoftscatapult-project-aims-speed-bing-search-improve-relevancy-194345
- [54] Microsoft. [n.d.]. Empowering Azure Storage with RDMA. Retrieved May 31, 2023 from https://www.microsoft.com/en-us/research/uploads/prod/2023/03/RDMA\_ Experience\_Paper\_TR-1.pdf
- [55] Bonaventura Del Monte, Steffen Zeuch, Tilmann Rabl, and Volker Markl. 2022. Rethinking Stateful Stream Processing with RDMA. In Processings of the International Conference on Management of Data, SIGMOD 2022. 1078–1092. https: //doi.org/10.1145/3514221.3517826
- [56] Abdullah Mueen, Suman Nath, and Jie Liu. 2010. Fast approximate correlation for massive time-series data. In Proceedings of the International Conference on Management of Data, SIGMOD 2010. 171–182. https://doi.org/10.1145/1807167.1807188
- [57] Brendan O'Connor. 2012. Cosine similarity, Pearson correlation, and OLS coefficients. Retrieved May 12, 2023 from https://brenocon.com/blog/2012/03/cosinesimilarity-pearson-correlation-and-ols-coefficients/
- [58] Vladimir Andrei Olteanu, Haggai Eran, Dragos Dumitrescu, Adrian Popa, Cristi Baciu, Mark Silberstein, Georgios Nikolaidis, Mark Handley, and Costin Raiciu. 2022. An edge-queued datagram service for all datacenter traffic. In 19th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2022. 761–777. https://www.usenix.org/conference/nsdi22/presentation/olteanu
- [59] Oracle. 2017. Oracle Exadata Architecture. Retrieved May 12, 2023 from https://www.oracle.com/database/technologies/exadata/architecture/
- [60] Thinh Hung Pham, Suhaib A. Fahmy, and Ian Vince McLoughlin. 2013. Low-Power Correlation for IEEE 802.16 OFDM Synchronization on FPGA. IEEE Trans. Very Large Scale Integr. Syst. 21, 8 (2013), 1549–1553. https://doi.org/10.1109/ TVLSI.2012.2210917
- [61] PyTorch. [n.d.]. TORCH.CORRCOEF. Retrieved May 17, 2023 from https: //pytorch.org/docs/stable/generated/torch.corrcoef.html
- [62] Waleed Reda, Marco Canini, Dejan Kostic, and Simon Peter. 2022. RDMA is Turing complete, we just did not know it yet!. In 19th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2022. 71–85. https: //www.usenix.org/conference/nsdi22/presentation/reda
- [63] Joseph Lee Rodgers and W Alan Nicewander. 1988. Thirteen ways to look at the correlation coefficient. American statistician 42 (1988), 59–66. https: //doi.org/10.2307/2685263
- [64] André Ryser, Alberto Lerner, Alex Forencich, and Philippe Cudré-Mauroux. 2022. D-RDMA: Bringing Zero-Copy RDMA to Database Systems. In 12th

Conference on Innovative Data Systems Research, CIDR 2022. www.cidrdb.org. https://www.cidrdb.org/cidr2022/papers/p77-ryser.pdf

- [65] Md. Shiblee Sadik, Le Gruenwald, and Eleazar Leal. 2018. Wadjet: Finding Outliers in Multiple Multi-Dimensional Heterogeneous Data Streams. In 34th IEEE International Conference on Data Engineering, ICDE 2018. 1232–1235. https: //doi.org/10.1109/ICDE.2018.00118
- [66] Yasushi Sakurai, Spiros Papadimitriou, and Christos Faloutsos. 2005. BRAID: Stream Mining through Group Lag Correlations. In Proceedings of the International Conference on Management of Data, SIGMOD 2005. 599–610. https: //doi.org/10.1145/1066157.1066226
- [67] David Sidler, Zeke Wang, Monica Chiosa, Amit Kulkarni, and Gustavo Alonso. 2020. StRoM: Smart Remote Memory. In *Fifteenth EuroSys Conference, EuroSys* 2020. 29:1–29:16. https://doi.org/10.1145/3342195.3387519
- [68] Snowflake. [n.d.]. Overview of Warehouses. Retrieved May 31, 2023 from https://docs.snowflake.com/en/user-guide/warehouses-overview
- [69] Snowflake. 2014. Understanding Snowflake Table Structures. Retrieved May 12, 2023 from https://docs.snowflake.com/en/user-guide/tables-micro-partitions
- [70] NCSS Statistical Software. [n.d.]. Correlation Matrix. NCSS. https://ncsswpengine.netdna-ssl.com/wp-content/themes/ncss/pdf/Procedures/NCSS/ Correlation\_Matrix.pdf
- [71] Statistics Solutions. [n.d.]. Correlation (Pearson, Kendall, Spearman). Statistics Solutions. Retrieved March 23, 2022 from https://www.statisticssolutions.com/freeresources/directory-of-statistical-analyses/correlation-pearson-kendallspearman/
- [72] Laerd Statistics. [n.d.]. Pearson Product-Moment Correlation. Laerd Statistics. Retrieved March 23, 2022 from https://statistics.laerd.com/statisticalguides/pearson-correlation-coefficient-statistical-guide.php
- [73] ETH Zurich Systems Group. [n.d.]. Scalable Network Stack for FPGAs (TCP/IP, Ro-CEv2). Retrieved November 03, 2022 from https://github.com/fpgasystems/fpganetwork-stack
- [74] ETH Zurich Systems Group. 2020. Heterogeneous Accelerated Compute Cluster. Retrieved November 03, 2022 from https://systems.ethz.ch/research/dataprocessing-on-modern-hardware/hacc.html
- [75] ETH Zurich Systems Group. 2023. AMNES. Retrieved August 10, 2023 from https://github.com/fpgasystems/amnes
- [76] Qing Wang, Youyou Lu, Jing Wang, and Jiwu Shu. 2022. Replicating Persistent Memory Key-Value Stores with Efficient RDMA Abstraction. CoRR abs/2209.09459 (2022). https://doi.org/10.48550/arXiv.2209.09459 arXiv:2209.09459
- [77] Yingjun Wu, Jia Yu, Yuanyuan Tian, Richard Sidle, and Ronald Barber. 2019. Designing Succinct Secondary Indexing Mechanism by Exploiting Column Correlations. In Proceedings of the International Conference on Management of Data, SIGMOD 2019. 1223–1240. https://doi.org/10.1145/3299869.3319861
- [78] AMD Xilinx. 2022. DMA/Bridge Subsystem for PCI Express Product Guide (PG195). AMD Xilinx. https://docs.xilinx.com/r/en-US/pg195-pcie-dma
- [79] AMD Xilinx. 2022. Overview of Arbitrary Precision Integer Data Types (UG1399). AMD Xilinx. https://docs.xilinx.com/r/en-US/ug1399-vitis-hls/Overview-of-Arbitrary-Precision-Integer-Data-Types
- [80] Hui Xiong, Shashi Shekhar, Pang-Ning Tan, and Vipin Kumar. 2006. TAPER: A Two-Step Approach for All-Strong-Pairs Correlation Query in Large Databases. *IEEE Trans. Knowl. Data Eng.* 18, 4 (2006), 493–508. https://doi.org/10.1109/ TKDE.2006.1599388
- [81] Lei Yu and Huan Liu. 2003. Feature Selection for High-Dimensional Data: A Fast Correlation-Based Filter Solution. In Machine Learning, Proceedings of the Twentieth International Conference (ICML) 2003. 856–863. http://www.aaai.org/ Library/ICML/2003/icml03-111.php
- [82] Erfan Zamanian, Xiangyao Yu, Michael Stonebraker, and Tim Kraska. 2019. Rethinking Database High Availability with RDMA Networks. Proc. VLDB Endow. 12, 11 (2019), 1637–1650. https://doi.org/10.14778/3342263.3342639
- [83] Jian Zhang and Joan Feigenbaum. 2006. Finding highly correlated pairs efficiently with powerful pruning. In Proceedings of the International Conference on Information and Knowledge Management, CIKM 2006. 152–161. https://doi.org/10.1145/1183614.1183640
- [84] Tao Zhang, Tianqing Zhu, Ping Xiong, Huan Huo, Zahir Tari, and Wanlei Zhou. 2020. Correlated Differential Privacy: Feature Selection in Machine Learning. *IEEE Trans. Ind. Informatics* 16, 3 (2020), 2115–2124. https://doi.org/10.1109/TII. 2019.2936825
- [85] Yunyue Zhu and Dennis E. Shasha. 2002. StatStream: Statistical Monitoring of Thousands of Data Streams in Real Time. In Proceedings of 28th International Conference on Very Large Data Bases, VLDB 2002. Morgan Kaufmann, 358–369. https://doi.org/10.1016/B978-155860869-6/50039-1