2024 IEEE International Parallel and Distributed Processing Symposium (IPDPS) | 979-8-3503-8711-7/24/$31.00 ©2024 IEEE | DOI: 10.1109/IPDPS57955.2024.00040

Accelerating Lossy and Lossless Compression on Emerging BlueField DPU Architectures

Yuke Li†, Arjun Kashyap†, Weicong Chen†, Yanfei Guo‡, and Xiaoyi Lu†*
†University of California, Merced, Merced, CA, USA
{yli304, akashyap5, wchen97, xiaoyi.lu}@ucmerced.edu
‡Argonne National Laboratory, Lemont, IL, USA
yguo@anl.gov

***Abstract*—Data compression has become a crucial technique in addressing performance bottlenecks caused by increasing data volumes in High-Performance Computing (HPC), Big Data, and Deep Learning (DL). Despite its potential to boost system performance, recent studies have identified significant challenges with existing compression methods, mainly due to their high computational demands amidst continuously growing data sizes. Concurrently, the advent of Data Processing Units (DPUs), equipped with programmable System-on-Chip (SoC) and specialized compression accelerators, offers a promising opportunity to alter the landscape of data compression. This paper explores the complexities and potential of leveraging NVIDIA BlueField DPUs to accelerate lossy and lossless compression. Towards this, we introduce *PEDAL*, an innovative library that leverages the hardware capabilities of DPUs to unify and optimize data compression designs. Moreover, we seamlessly co-design PEDAL with the popular MPICH MPI library, demonstrating up to 101x speedup in compression time and 88x decrease in communication latency. Drawing on these achievements, we share our experience with various research communities about accelerating data compression on DPUs in communication-oriented HPC scenarios.**

***Index Terms*—Data Processing Unit (DPU), BlueField DPU, Lossy Compression, Lossless Compression**

I. Introduction

Data compression is a technique aimed at effectively reducing the size of the data while maintaining its integrity or quality to the desired extent. It serves as a fundamental approach to addressing performance bottlenecks in the realms of parallel and distributed computing, where scientific and deep learning (DL) applications can benefit from elevated efficiency of data storage and movement [1]–[9].

Recent advancements in lossy and lossless data compression highlight the intense computational demands required to manage growing data volumes amidst the rise of Big Data, HPC, and DL technologies. To improve compression algorithm efficiency, strategies targeting various computing platforms like x86 CPUs [10]–[12], GPUs [13], [14], and FPGAs [15], [16] have been proposed, indicating a broad effort to address these computational challenges across mainstream hardware.

* is the corresponding author.

A. Motivation

In this evolving landscape, the emergence of Data Processing Units (DPUs), particularly NVIDIA's BlueField series [17], [18], heralds a fresh direction for data compression research. DPUs enhance the functionalities of traditional network interface cards (NICs) with general-purpose computing capabilities and dedicated hardware accelerators for compression algorithms, offering a novel adjunct to performing data compression on conventional platforms. As DPUs offer inherent proximity to the data transmission pathways, we believe DPUs present a unique advantage for crafting efficient data compression methodologies, especially for communication-oriented application scenarios within the HPC domain.

While our earlier studies [19], [20] have initiated the exploration of characterizing lossy and lossless compression and decompression performance on DPUs, **to the best of our knowledge, no comprehensive investigation has yet formulated a unified and efficient strategy for accelerating both lossy and lossless compression and decompression on both NVIDIA BlueField-2 and BlueField-3 DPUs.** Moreover, there is a notable absence of existing studies on DPU-based compression tailored for communication-oriented scenarios in the HPC domain. This notable absence thus forms a critical motivation for our inquiry, underlining the importance of our research efforts in this area.

B. Challenges

Yet, leveraging DPUs for lossless and lossy compression in communication-oriented workloads presents significant obstacles. We highlight three primary challenges that necessitate creative solutions and careful planning.

(1) Hardware Limitation in BlueField DPUs: The BlueField DPUs from NVIDIA inherently support a limited set of lossless compression algorithms through their specialized hardware compression engine (namely C-Engine), as documented in [21]. Furthermore, the support for efficient lossy compression on BlueField DPUs remains unclear. This opens up a promising avenue for research, prompting the question: *How can the SoC and C-Engine of different BlueField generations*

1530-2075/24/$31.00 ©2024 IEEE
DOI 10.1109/IPDPS57955.2024.00040

be harnessed to benefit both lossy and lossless compression during communication?

(2) Limitations of Existing Compression Designs: State-of-the-art compression designs, such as DEFLATE [22], zlib [23], SZ3 [12], SZX [13], cuSZ [14], and WaveSZ [15], primarily focus on compressing datasets for storage scenarios. None of these prior works have systematically evaluated both lossy and lossless compression and decompression performance, specifically in transmitting datasets with NVIDIA BlueField-2 and BlueField-3 architectures. However, efficient compression designs for communication are crucial, as the movement of large data volumes constitutes a major performance bottleneck in many HPC and Deep Learning (DL) applications [1], [2], [4], [24]–[30]. Consequently, there is an under-explored research challenge: *How can we leverage DPUs to facilitate on-the-fly lossy and lossless compression for communication-oriented workloads?*

(3) Complexity in Integrating Compression Designs with Communication Libraries: Enhancing performance through compression designs on DPUs for prominent communication libraries, such as Message Passing Interface (MPI), is a complex task. Moreover, achieving efficient integration while minimizing changes to the communication library's APIs is critical to ensure wide adoption. This leads us to the question: *How can we efficiently co-design the proposed lossy and lossless compression schemes with MPI to achieve high performance and minimal disruptions to the library's core functionality?*

C. Contributions

In response to these challenges, this paper makes the following contributions.

Firstly, we investigate and harness the SoC and the C-Engine embedded in BlueField-2 and BlueField-3. These components become the backbone of our innovative approach, where multiple compression and decompression algorithms find a home. Through this method, we enhance the efficiency of our proposed *PEDAL* library, seamlessly integrating both lossy (SZ3 [12]) and lossless (zlib [23], DEFLATE [22], and LZ4 [31]) compression schemes on these emerging devices. To optimize these processes, we introduce a repertoire of innovative techniques. These include extracting C-Engine initialization and intertwining it with MPICH [32] runtime, employing a streamlined header design to synchronize compression and decompression between senders and receivers, caching memory buffers for efficient reuse, and extending existing compression schemes to harness the combined capabilities of DPU's SoC and C-Engine. The comprehensive framework of these techniques is detailed in Section III.

Secondly, we incorporate our compression framework into a co-design with MPICH, focusing on the `MPI_Send` and `MPI_Recv` routines. This endeavor enables applications and upper-layer communication libraries to effortlessly execute efficient lossy or lossless compression and decompression. The significant performance improvements from this co-design are detailed in Section V, showcasing PEDAL's exceptional capabilities. Notable achievements include a 101-fold increase in compression efficiency and an 88-fold decrease in communication latency, marking PEDAL as a transformative element in enhancing data compression and communication speed.

Lastly, we share insights on accelerating lossy and lossless compression on BlueField DPUs across multiple communities, advocating for the re-integration of C-Engine in future DPU architectures and encouraging the DPU community to enhance algorithmic diversity and programmability. We suggest the MPI community evaluate strategies for optimizing data compression tasks offloaded to DPUs, highlighting the need for improved compression infrastructure within MPI frameworks. For PEDAL users, we emphasize the seamless integration of our co-design with MPICH, maintaining the MPI interface's consistency. Additionally, we offer the data compression community a foundation for advancing DPU-based compression designs, suggesting collaborative efforts for further performance enhancement.

II. Background

A. NVIDIA BlueField DPUs

Data Processing Units (DPUs), or Smart Network Interface Cards (SmartNICs), represent programmable network interface components designed to assist servers by offloading computational tasks. Our study employs the following two DPUs that span two generations of NVIDIA BlueField DPU series:

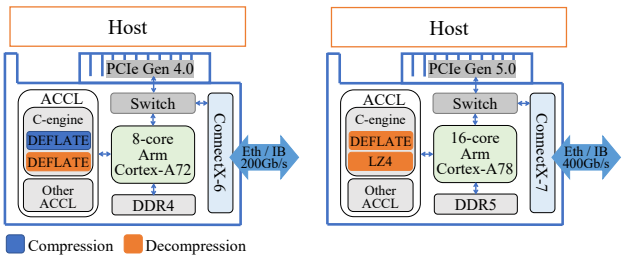
NVIDIA BlueField-2: As illustrated in Figure 1a, the NVIDIA BlueField-2 DPU features hardware accelerators for offloading key networking, storage, and security tasks from the CPU. It incorporates an ARM-based System-on-Chip architecture and DDR4 memory, paired with a PCIe switch and NVIDIA ConnectX-6 NIC, supporting up to 200 Gb/s Ethernet or InfiniBand connectivity.

NVIDIA BlueField-3: The BlueField-3 DPU represents a notable advancement over its predecessor. Illustrated in Figure 1b, it supports up to 400 Gb/s Ethernet or InfiniBand via the ConnectX-7 NIC. Upgrading to DDR5 standards with more channels enhances RAM throughput by up to 4.2x compared to BlueField-2. The CPU within the SoC also sees an upgrade to 16 ARM Cortex-A78 cores, doubling the core count and quadrupling computational power [33]. With the addition of AI/HPC accelerators, the BlueField-3 DPU sets new performance and efficiency benchmarks for handling complex data and communication-oriented tasks such as data compression.

BlueField DPUs can operate in two distinct modes:

Separated Host Mode: In this configuration, the System-on-Chip (SoC) of the DPU serves as an independent host analogous to external servers. The architecture ensures that both the host and the DPU's SoC are allocated with unique network addresses. This mode mirrors the functionality of off-path SmartNICs, with the host's traffic bypassing the SoC.

Embedded CPU Function Mode (SmartNIC Mode): Under this mode, the SoC oversees the NIC resources and data



(a) BlueField-2 DPU Arch. (b) BlueField-3 DPU Arch.

Fig. 1: BlueField DPUs architecture.

pathways. Therefore, the network traffic to and from the host is channeled through a virtual switch on the SoC cores. In the SmartNIC mode, the DPU acts as an on-path SmartNIC, where the SoC within the DPU is utilized to adjust network packets during data transmission.

Notably, SmartNIC mode results in losing RDMA-IB support on the host due to hardware limitations up to the latest BlueField devices [34], i.e., BlueField-3. Therefore, our data compression methodology in this paper builds upon the Separated Host Mode for DPUs.

B. Compression

Reducing data volumes through compression techniques has gained increasing attention, particularly in large-scale HPC applications. Within the data compression community, there are two primary research directions: lossy and lossless compression.

Lossy Compression: This group of algorithms yields high compression ratios by strategically omitting parts of the data. In application scenarios where reducing data volumes is paramount and loss of data integrity and accuracy are tolerable to some extent, lossy compression methods, such as SZ3 [12], prove indispensable. SZ3's compression sequence starts with a preprocessor that normalizes and conditions the data. Following this, a predictor utilizes algorithms to accurately forecast data values, effectively mitigating prediction errors. Next, the quantizer approximates these prediction errors, balancing precision retention with high compression ratios. An encoder then implements entropy coding. Finally, a lossless compressor is employed to ensure no additional data loss while maximizing compression efficiency.

Lossless Compression: This group of compression algorithms can fully ensure data integrity during the decompression process by trading off lower compression ratios compared to lossy compression. DEFLATE [22] exemplifies lossless compression by integrating LZ77 [35] to remove duplicate strings from data blocks and Huffman coding [36] for optimal symbol encoding. Specifically, LZ77 minimizes redundancy by substituting repeated sequences with references to earlier instances, and Huffman coding optimizes size reduction by allocating shorter codes to common symbols and longer codes to infrequent ones.

III. Proposed Designs for PEDAL

Inspired by the performance characterization presented in our prior studies [19], [20], we have developed a uniform compression and decompression library, PEDAL, to fully leverage the SoC and the data compression accelerator, namely C-Engine, on BlueField-2 and BlueField-3 for multiple compression and decompression designs. In the rest of this section, we will elaborate our designs towards this uniform compression and decompression library with the selected compression designs.

Furthermore, we have also attempted to optimize these designs by drawing inspiration from our prior studies on DPU compression characterization, aiming to further enhance the performance of these designs.

A. Selected Compression Algorithms

We select four prominent compression algorithms as outlined in Table I. The chosen lossless algorithms include DEFLATE [22], zlib [23], and LZ4 [31]. These algorithms are versatile to accommodate a range of data types such as text files, images, binary data, and databases. Concurrently, we also select one lossy design, SZ3 [12], which primarily targets scientific data compression.

TABLE I: Compression designs and features.

Algorithm	Purpose	Lossless	Lossy
DEFLATE	General Data Compression	✓	
zlib	General Data Compression	✓	
LZ4	General Data Compression	✓	
SZ3	Scientific Data Compression		✓

All selected algorithms were executed on the SoC. When leveraging the BlueField DPU's C-Engine, one must utilize the NVIDIA DOCA SDK [21]. This SDK enables hardware-accelerated compression and decompression for DEFLATE and LZ4. The compression designs supported by BlueField DPU's SoC and C-Engine are summarized in Table II. For the lossless compression algorithms, our experiments employed their default settings (e.g., compression levels and window sizes). For the lossy compression design (SZ3), an error bound of `1e-4` was employed, which is a common configuration in SZ3 and comparable to lossless compression designs.

TABLE II: Compression algorithms supported by different hardware on BlueField-2 (BF2) and BlueField-3 (BF3).

Algorithm	SoC	C-Engine	
		Compression	Decompression
DEFLATE	BF2, BF3	BF2	BF2, BF3
zlib	BF2, BF3	-	-
LZ4	BF2, BF3	-	BF3
SZ3	BF2, BF3	-	-

B. PEDAL Overview

PEDAL, our proposed uniform compression and decompression library, has been designed to maximize the utilization of both BlueField DPU's SoC and C-Engine for lossy and lossless compression algorithms. PEDAL encompasses three designs: 1) it optimizes zlib and SZ3, originally only running on the SoC, to leverage both the SoC and C-Engine; 2) it ports all compression algorithms in Table I to the SoC; and 3) it wraps the existing algorithms, i.e., DEFLATE and LZ4, provided by the C-Engine. Based on this, PEDAL can enumerate up to eight *compression designs*. The comprehensive architecture of PEDAL is showcased in Figure 2.

In the upper layer (blue enclosure), PEDAL provides streamlined APIs to compress and decompress a range of data buffers. By co-designing with MPICH (discussed in Section IV), the data buffer pointer can be passed to PEDAL compress and decompress APIs to execute the selected compression design. In the default case, different compression designs have different APIs, limiting broad-range applicability. Therefore, in the middle layer (green enclosure), we propose integrating various compression designs into a set of uniform APIs. Based on this goal, we port algorithms highlighted in yellow boxes to the SoC and wrap the default compression designs provided by C-Engine (highlighted in orange boxes) into the uniform PEDAL API.

In addition, we optimize the lossless zlib and lossy SZ3 by leveraging both the SoC and C-Engine, as highlighted in green boxes in zlib (C-Engine) and SZ3 (C-Engine), respectively. As a result, via the PEDAL_Compress and PEDAL_Decompress APIs, users can execute SZ3, DEFLATE, zlib, and LZ4 by combining strategies highlighted in three boxes, constituting up to eight compression designs that leverage both the SoC and C-Engine. We highlighted which compression algorithms can be deployed on which hardware architecture (i.e., the SoC or the C-Engine) or BlueField DPU generations in Table III.

Subsequent sections provide an in-depth exploration of PEDAL's intricacies, encompassing its compression and decompression strategies, the signature of PEDAL header, and its standardized APIs.

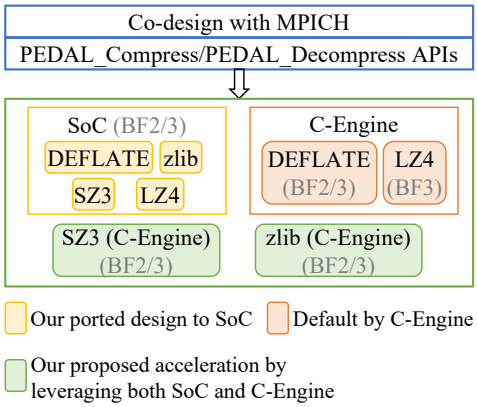


Fig. 2: Architecture overview of PEDAL.

TABLE III: Compression designs supported by the PEDAL on different hardware of BlueField-2 and BlueField-3. Compared with the Table II, we extend more compression designs and support on BF2/BF3, which are italicized in this table: zlib and SZ3 on C-Engine.

Algorithm	SoC Core	C-Engine	
		Compression	Decompression
DEFLATE	BF2, BF3	BF2	BF2, BF3
zlib	BF2, BF3	*BF2*	*BF2, BF3*
LZ4	BF2, BF3		BF3
SZ3	BF2, BF3	*BF2*	*BF2, BF3*

C. Compression Designs in PEDAL

In our prior DPU compression characterization efforts, we noted that the buffer preparation and the DOCA initialization process collectively consumed approximately 90% of the total execution time. This observation indicates that using C-Engine incurs significant overhead. To address this overhead, PEDAL prearranges all essential buffers through a memory pool, and sets up the DOCA configuration only at initialization. The memory pool is devised to reuse intermediate buffers, and eliminate the frequent need for memory allocation, deallocation, and mapping between regular and DOCA-operable memory during each compression and decompression execution.

1) Lossless Compression: PEDAL utilizes DOCA to extend the lossless compression design from the SoC to the C-Engine. DOCA supports DEFLATE compression and decompression operations on BlueField-2; and DEFLATE decompression and LZ4 decompression operations on BlueField-3. Apart from these lossless compression algorithms provided by DOCA, we modify the compression design for zlib to enable it to run on the C-Engine. This is because zlib uses DEFLATE as a compression algorithm under the hood, plus an extra header and a trailer combined with the compressed data. PEDAL assigns computation to the zlib header and trailer on the SoC, while diverting the actual data compression execution on the C-Engine.

In Figure 3, the compression steps for zlib involves: 1) `init_data_env`: Initialize the DOCA configurations and memory pool on the SoC; 2) `prepare_data_buffer`: Map the data into DOCA-specific buffer on the SoC; 3) `data_compressing`: Submit a compression job via DOCA to the C-Engine, where the data is scheduled to be compressed. After the compression job is complete, the compressed data will be stored in a user-specified buffer on the SoC; and 4) `zlib_header` and `zlib_trailer`: Calculate zlib-specific header and trailer on the SoC, and put them to the head and tail of the compressed data.

2) Lossy Compression: The modules designed for lossy compression are illustrated in Figure 4. Since lossy compression algorithms involve lossless compression, we identify a potential opportunity to accelerate the original design, where PEDAL utilizes the C-Engine to perform lossless compression within a lossy compression process. To exemplify, we detect that SZ3 utilizes DEFLATE or LZ4 within its compression

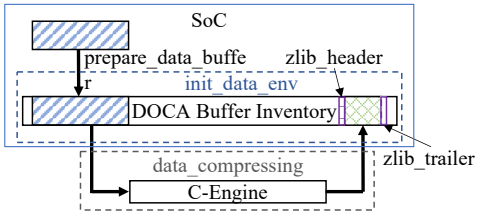


Fig. 3: The optimized zlib by leveraging both SoC and C-Engine.

process. Therefore, PEDAL can execute DEFLATE using C-Engine to accelerate SZ3.

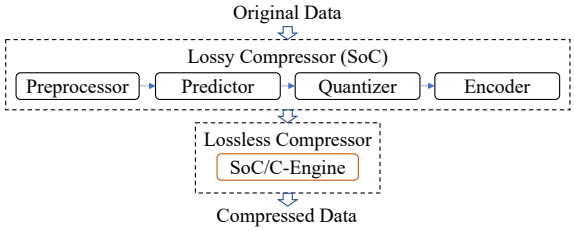


Fig. 4: Lossy compression pipeline in PEDAL-optimized SZ3.

D. Unified APIs

Listing 1 illustrates the streamlined APIs offered by PEDAL. `PEDAL_Init` establishes the PEDAL context and environment, including the memory allocation, deallocation, mapping between regular and DOCA-operable memory, and DOCA initialization. By doing so, we can integrate the `PEDAL_Init` function into the communication initialization part of the communication runtime library, such as `MPI_Init` . Then, users can utilize `PEDAL_compress` and `PEDAL_decompress` for efficient data compression and decompression with selected compression designs. To fully leverage the PEDAL library, it's crucial to focus on making the API robust and user-friendly. This ensures it works well for a variety of hardware, whether running on the SoC, the C-Engine, or both. For instance, `PEDAL_compress` processes a data buffer, outputs a new buffer address with compressed data, and updates `out_count` to indicate the compressed data size. The `datatype` parameter aids in lossy compression by specifying the algorithm's required datatype (e.g., integer, float, or double), enhancing compression efficiency.

```
/* PEDAL context and environment initialization */
int PEDAL_init(void *user_ctx);

/* message compression by PEDAL */
void *PEDAL_compress(int datatype, const void *in,
    int count, int *out_count);

/* message decompression by PEDAL */
void PEDAL_decompress(int datatype, void *in, int
    in_count, void *in_out_buf, int in_out_count);

/* Finalize PEDAL environment */
int PEDAL_finalize(void *user_ctx);
```

Listing 1: The uniformed PEDAL APIs for data compression and decompression.

PEDAL allows flexible selection of the proposed compression designs. As depicted in Table III, while all compression algorithms can run on the SoC, compatibility with BlueField DPUs' C-Engine varies by generation. PEDAL can automatically detect the hardware capability of the BlueField series to determine supported compression designs, and intelligently fall back to SoC-based compression designs if a compression algorithm is unsupported by the C-Engine, thus avoiding software failures.

E. PEDAL Header

As shown in Figure 5, on the sender side, the original data buffer pointer will be passed to PEDAL compression. Within the PEDAL compression, two data types are populated: a tiny header and the compressed data itself. The header comprises three bytes, with the first and third bytes acting as indicators in `0xFF` to signal whether the data received is compressed.

Additionally, these indicators assist in determining the appropriate compression design for decompressing on the receiver side. The number in the second byte, identified as `AlgoID`, specifies the selected compression design. This header, along with the compressed data, is then transmitted to the receiver. Upon receiving the data, the receiver's PEDAL decompression can determine whether the data is compressed and, if so, identify the specific compression design used. Subsequently, it applies the corresponding design to decompress the data.

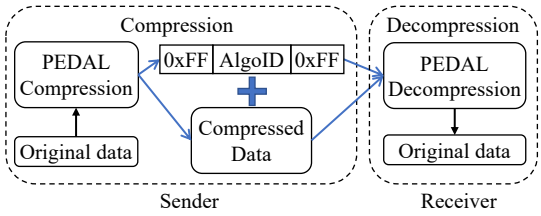


Fig. 5: Overview of the tiny PEDAL header design.

IV. Co-design with MPICH: A Pluggable Approach with Code Generation

This section outlines our objective of supporting on-the-fly compression using the MPI communication runtime library.

Figure 6 illustrates how we integrate PEDAL with MPICH, one of the most widely used MPI implementations in the HPC domain. On the sender side, PEDAL sits between the shim (to bridge the MPI abstraction and lower-level hardware interfaces) and transport layers (i.e., UCX [37]/OFI [38]). When a message requires compression, PEDAL is invoked internally to compress the data and generate a new buffer that consists of the PEDAL header and the compressed message. This new buffer is then forwarded to the transport layer (UCX/OFI) and transmitted across the network to the receiver side. On the receiver side, PEDAL is integrated within the MPICH's binding layer, wherein it calls into the MPICH's internal binding function [39]. Instead of using a user-defined

buffer from `MPI_Recv`, MPICH posts the receive request with a PEDAL-generated buffer. Once the complete message arrives in the PEDAL buffer, PEDAL decompresses it, and the resultant decompressed message is populated into the user-defined buffer directly without an additional copy.

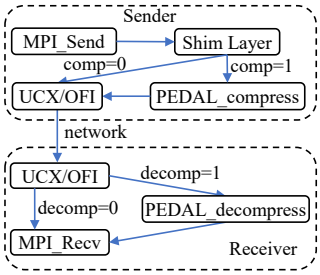


Fig. 6: Overall architecture for co-design with MPICH.

This co-design approach has two primary benefits: flexibility in embracing future compression designs and the ability to choose whether to employ compression designs in the user-built MPICH library. For instance, future developments could involve various compression designs using the SoC and C-Engine to achieve parallel compression and decompression. Users only need to update PEDAL to support additional compression designs, eliminating the need to rebuild the entire MPICH library. Moreover, the streamlined PEDAL APIs provide flexibility during the MPICH code-building process, allowing users to decide whether to include PEDAL when building MPICH on their machines. This flexible approach caters to diverse performance requirements and preferences while ensuring extensibility and customization.

In addition to the co-design implementation within the `MPI_Send` and `MPI_Recv`, the `PEDAL_init` is integrated into the `MPI_Init` to minimize the overhead associated with PEDAL initialization.

Currently, PEDAL operates on MPI's Rendezvous (RNDV) protocol for larger message sizes rather than the Eager protocol for smaller message sizes. This is mainly due to the latency overhead of compression and decompression operation, which prevent compression techniques from benefiting short messages. At present, both PEDAL and MPICH run on the DPU to minimize the data movement overhead across the PCIe bus.

V. Evaluation

A. Dataset

Alongside the compression designs, we selected eight widely-used datasets with various sizes for benchmarking as shown in Table IV. For lossless designs, we chose five datasets from multiple fields, from floating point datasets to XML files. For lossy compression designs, we chose three datasets with scientific data from SDRBench [40], [41]. We then describe the datasets in detail.

silesia [42]: The silesia corpus is a benchmark collection for testing lossless data compression algorithms, designed to address limitations in existing standard corpora. It features files ranging from 6 MB to 51 MB and covers diverse data types. In silesia corpus, we choose `xml`, `mr`, `samba` and `mozilla`. Notably, it includes database files, concatenated large projects, executable files, and non-compressible medical images. It also introduces various text file formats like PDF, HTML, and XML. It offers a more relevant and up-to-date evaluation dataset for compression algorithms.

obs_error [43]: obs is a scientific IEEE standard floating-point dataset, and **obs_error** specifies the brightness temperature errors from a weather satellite. The high-precision floating point format and scientific computing characteristics yield a high-quality dataset for benchmarking the compression designs.

exaalt [40], [41]: is a molecular dynamics simulation dataset that involves intensive floating points.

TABLE IV: Eight datasets with various sizes and features.

Design	Dataset	Description	Size (MB)	Hatches in Fig. 7–9
Lossless	silesia/xml	XML files, text	5.1	
	silesia/mr	3-D MRI image, DICOM	9.51	
	silesia/samba	source code and graphics	20.61	
	obs_error	single Float-Point	30	
	silesia/mozilla	exe	48.85	
Lossy	exaalt-dataset1	MD simulation, single float-point	10	
	exaalt-dataset3		31	
	exaalt-dataset2		64	

B. Testbeds

Evaluations for BlueField-2 and BlueField-3 were conducted on the Thor cluster, which belongs to the HPC Advisory Council High-Performance Center (HPCAC) [44]. The BlueField-2 DPU within this cluster has 8 ARM Cortex-A72 2.75 GHz cores, accompanied by 16 GB of on-board DDR4 DRAM. The BlueField-3 boasts 16 ARM Cortex-A78 cores and 16 GB of on-board DDR5 memory. Both DPUs operate under Rocky Linux 9.1 and are equipped with the DOCA SDK v2.0. Throughout our experiments, we ensure that the BlueField-2/3 DPUs operate in the Separated Host mode.

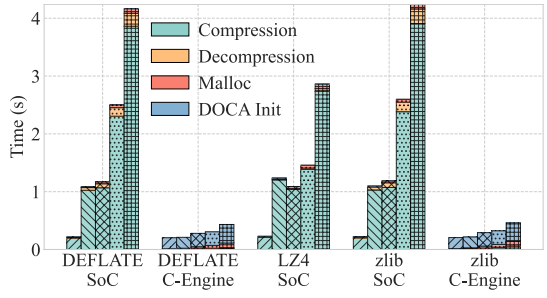
C. Evaluation on Compression Designs Supported by PEDAL

1) Lossless Compression Evaluation with PEDAL: Figure 7 provides a detailed analysis of the time distribution across four fractions (including DOCA initialization, buffer preparation, data compression, and decompression) engaged in lossless compression designs on BlueField DPUs on five datasets as enumerated in Table IV. The fundamental stages include memory buffer allocation for compressed and decompressed data, the initialization of the C-Engine, especially if DOCA is engaged, and the primary task of compressing and decompressing the data itself.

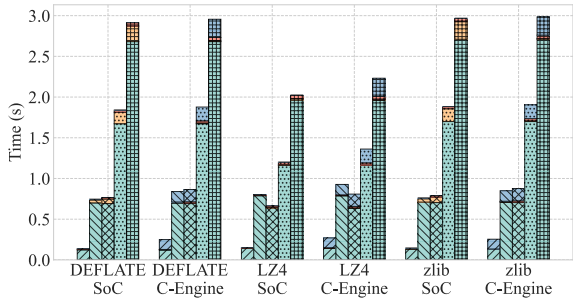
Figure 7a indicates the time distribution of the entire lossless compression and decompression execution on the SoC and C-Engine for BlueField-2. Harnessing the C-Engine of BlueField-2 results in a noteworthy reduction in the total

compression and decompression time, with an acceleration factor reaching 9.67× for lossless compression designs. However, when running compressing or decompressing through C-Engine, it is evident that the initialization stage (`DOCA_Init`) along with memory allocation contribute to approximately 94% of the total execution time, particularly when handling smaller message sizes, around 5.1 MB.

Figure 7b illustrates the time distribution of compression and decompression on the SoC and C-Engine for BlueField-3 DPU. In marked contrast to BlueField-2, the C-Engine of BlueField-3 exhibits comparable, and in certain instances, marginally prolonged time consumption in relation to its SoC. Delving deeper into this observation, it is ascertained that the C-Engine quipped on BlueField-3 cannot facilitate compression operations. Instead, it exclusively supports decompression functionalities. Consequently, any compression designs initiated on BlueField-3's C-Engine are automatically redirected to execute on the SoC. While the C-Engine is proficient in delivering expedited decompression time, the overhead engendered by the DOCA initialization mitigates the performance advantages conferred by the C-Engine. Consequently, the aggregate time required for compression and decompression remains relatively consistent across different lossless compression designs, regardless of whether they are executed on BlueField-3's SoC or C-Engine.



(a) BlueField-2



(b) BlueField-3

Fig. 7: Time distribution for lossless compression designs on BlueField-2 and BlueField-3 across varied datasets. Within each compression design, datasets are arranged in ascending order of size from left to right.

In Figure 7, one can observe that the potential performance advantages of BlueField's SoC and its C-Engine are somewhat masked by the pronounced overheads associated with memory allocation and DOCA initialization. PEDAL addresses this by sidestepping these overheads at runtime; it delegates memory allocation and engine initialization tasks to the `PEDAL_INIT` phase. This strategic approach empowers upper-layer communication protocols, such as MPI communication, to capitalize on the pure compression and decompression capabilities of BlueField DPUs, as elaborated in SectionsV-D and V-E.

Figure 8 offers a comparative perspective on the raw performance of PEDAL's lossless compression and decompression between BlueField-2 and BlueField-3 DPUs, across five distinct datasets as enumerated in Table IV. Three insights emerge from this comparison: 1) regardless of the computation platforms, whether the SoC or the C-Engine, there is a direct correlation between the compression and decompression times and the dataset size; 2) the decompression time is invariably shorter than the compression time on both BlueField-2 and BlueField-3 DPUs; and 3) the C-Engine consistently exhibits superior speed in handling any lossless compression technique (be it DEFLATE, LZ4, or zlib) compared to the SoC for DPUs.

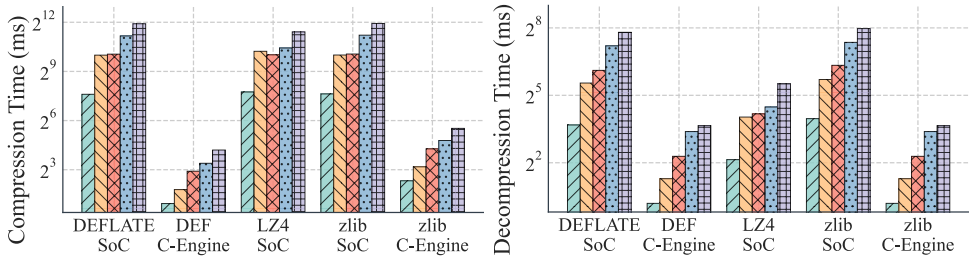
For instance, the C-Engine for BlueField-2 is remarkably 101.8× and 11.2× swifter than its SoC counterparts when processing the DEFLATE compression and decompression design with the 5.1 MB silesia/xml dataset. For the zlib compression design with the 48.84 MB silesia/mozilla dataset, the compression operation on the C-Engine is 84.6× of on the SoC, the decompression operation on the C-Engine is 20× of on the SoC for BlueField-2. Compared with the C-Engine for BlueField-2 and BlueField-3, the BF3's C-Engine outperforms BF2's 1.78× and 1.28× on DEFLATE decompression operation with dataset sizes 5.1 MB and 48.84 MB, respectively.

Consequently, to maximize efficiency, PEDAL predominantly relies on the C-Engine of BlueField (when applicable) over the SoC, thus ensuring optimal compression and decompression performance.

2) Lossy Compression Evaluation with PEDAL: Figure 9 indicates the time distribution across four fractions engaged in lossy compression designs on BF2/BF3 on three datasets ranging from 10 MB to 64 MB as enumerated in Table IV.

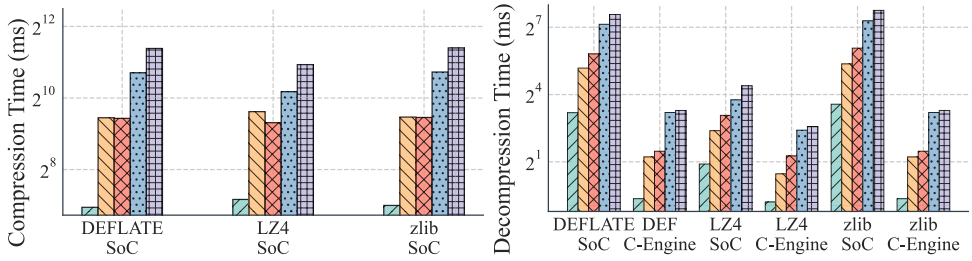
Interestingly, BF2 exhibits comparable cumulative execution times for compression and decompression across both the SoC and the C-Engine. This phenomenon can be attributed to the architecture of SZ3, where the consumption of time by the lossless compression component does not significantly impact SZ3's performance-critical path. As a result, leveraging the C-Engine exclusively for its lossless compression doesn't markedly amplify the speed compared to using the SoC. However, this highlights a prospective hybrid design avenue for exploiting both SoC and C-Engine in parallel, especially when the SoC is occupied. Importantly, leveraging the C-Engine does not detrimentally affect the performance metrics of lossy compression and decompression processes on BF2.

Transitioning to BlueField-3, an observable distinction emerges. When employing the SoC for lossy compression techniques, performance improvements are up to 1.58× compared to employing the C-Engine for a 10 MB dataset. The



(a) BlueField-2, Compression.

(b) BlueField-2, Decompression.



(c) BlueField-3, Compression.

(d) BlueField-3, Decompression.

Fig. 8: Compression and Decompression time across varied datasets. Within each compression design, datasets are arranged in ascending order of size from left to right.

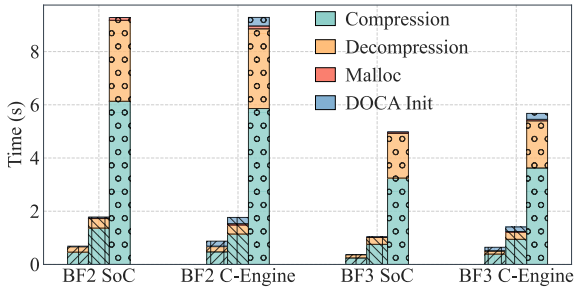


Fig. 9: Time distribution for lossy compression designs with SZ3 running on the BlueField-2/3.

underpinning reason for the relatively diminished speed of the BF3 C-Engine compared with the BF3 SoC in compression tasks lies in the BF3's limitation – it lacks native support for C-Engine-accelerated compression operation. Consequently, operations redirect to the SoC DEFLATE design. Since the DEFLATE design is less optimized than SZ3's inherent zstandard compressor in compression latency, the compression speed on BF3's C-Engine lags behind that of its SoC counterpart. Notably, on BlueField-3, the decompression of datasets subjected to lossy compression techniques consistently outperforms their counterparts. This behavior mirrors the trends previously discerned during the exploration of lossless compression and decompression dynamics on DPUs.

3) Compression Ratio Evaluation: The efficacy of a compression algorithm is often gauged not just by its speed but also by its ability to reduce data sizes while maintaining its integrity or quality to the desired extent. Table V enumerates the compression ratios achieved across various datasets on both lossless and lossy compression designs. The compression ratios are calculated by dividing the size of the original data size by the compressed data size (the larger the value, the smaller the size of the compressed data).

Analyzing both the compression ratio and the compression time shown above reveals that dataset size or compression ratio alone cannot reliably predict compression time efficiency. For instance, the `silesia/mr` dataset, at 9.51 MB, is smaller than the `silesia/samba` dataset, which is 20.61 MB, yet it shows a lower compression ratio across all tests. However, both datasets demonstrate similar compression times on both BF2 and BF3's SoC, as illustrated in Figure 8.

TABLE V: Compression Ratio of compression designs supported by PEDAL.

Dataset	DEFLATE	LZ4	zlib
obs_error	1.469	1.204	1.469
silesia/mozilla	2.683	2.319	2.683
silesia/mr	2.712	2.348	2.712
silesia/samba	3.963	3.517	3.963
silesia/xml	7.769	6.933	7.769

(a) Datasets used for lossless compression design.

Dataset	SZ3	SZ3(C-Engine)
exaalt-dataset1	2.941	2.940
exaalt-dataset3	5.745	5.844
exaalt-dataset2	5.378	4.971

(b) Datasets used for lossy compression design.

Insights From PEDAL Compression Designs

Performance Overheads: Overheads from memory allocation and engine initialization achieve up to 94% of total execution time, especially for smaller datasets like 5.1 MB.
PEDAL's Efficiency: By tackling overheads during its initialization, PEDAL showcases efficiency, especially evident in BlueField-2's compression acceleration of up to 9.67×.
BlueField-2 vs. BlueField-3: While BF3's C-Engine excels than BF2's C-Engine in DEFLATE decompression operation 1.78× with dataset 5.1 MB, BF3 lacks of C-Engine support in compression operation.
Lossy Compression: On BF2, compression/decompression times remain consistent across components. For BF3, when handling lossy compression for a 10 MB dataset, the SoC designs are up to 1.58× faster than its C-Engine design due to the latter's reliance on the slower DEFLATE design.

D. Evaluation on MPI Point-to-Point Micro Benchmarks

This section assesses PEDAL's capability in facilitating on-the-fly compression and decompression for MPI point-to-point communication. We benchmark the latency introduced by various compression designs, contrasting the native compression algorithms against the optimized designs offered by PEDAL.

Figure 10 delineates the performance metrics for MPI's point-to-point operations as gauged by the OSU Micro-Benchmarks [45]. This is executed across various message sizes, encompassing lossy and lossless compression configurations. For our comparative analysis, the baseline design epitomizes the compression and decompression duration on the BlueField-2 platform without PEDAL. Notably, this baseline does not engage the PEDAL framework, implying that memory allocation and the DOCA initialization procedure are invoked during every message transmission.

Lossless Compression: For PEDAL configurations on BlueField-3 that utilize the SoC, specifically SoC_DEFLATE, SoC_LZ4, and SoC_zlib, we observe a reduction of communication time by up to 40% compared with those on BlueField-2. This enhanced efficiency can be attributed to the superior computational prowess of BlueField-3's SoC core compared to its BlueField-2 counterpart. Meanwhile, PEDAL's design based on C-Engine demonstrate an acceleration of up to 88× relative to the baseline on BlueField-2 for DEFLATE and zlib methodologies. This substantial performance boost stems from PEDAL's design which migrates overheads from the runtime to the initialization phase. However, it is noteworthy that BlueField-3's C-Engine exhibited elongated communication times for DEFLATE and zlib methods, surpassing even the baseline. This limitation arises due to the C-Engine's inability to support certain compression operations, a detail elucidated in Section V-C. BlueField-2, with its lack of support for LZ4 on its C-Engine, consequently relegates LZ4 compression to the SoC core, leading to suboptimal performance.

Lossy Compression: Figure 10f showcases the communication time when deploying the SZ3 lossy compression design. With the PEDAL, both BlueField-2 and BlueField-3 manifested a substantial decrease in latency, recording improvements of up to 47.3% and 48% respectively over the baseline. This is attributed to the architectural design of PEDAL combined with the robustness of the SoC.

Insights From PEDAL-Optimized MPI Point-to-Point Communication

Baseline: Without PEDAL, every transmission requires repetitive memory allocation and `DOCA_Init` procedure.
Lossless Compression: BF3's SoC achieved up to a 40% reduction of BF2's in communication time; PEDAL attained up to 88× speedup over the baseline by moving overheads from runtime to initialization.
Lossy Compression: PEDAL yielded latency reductions up to 47.3% on BF2 and 48% on BF3 over the baseline.

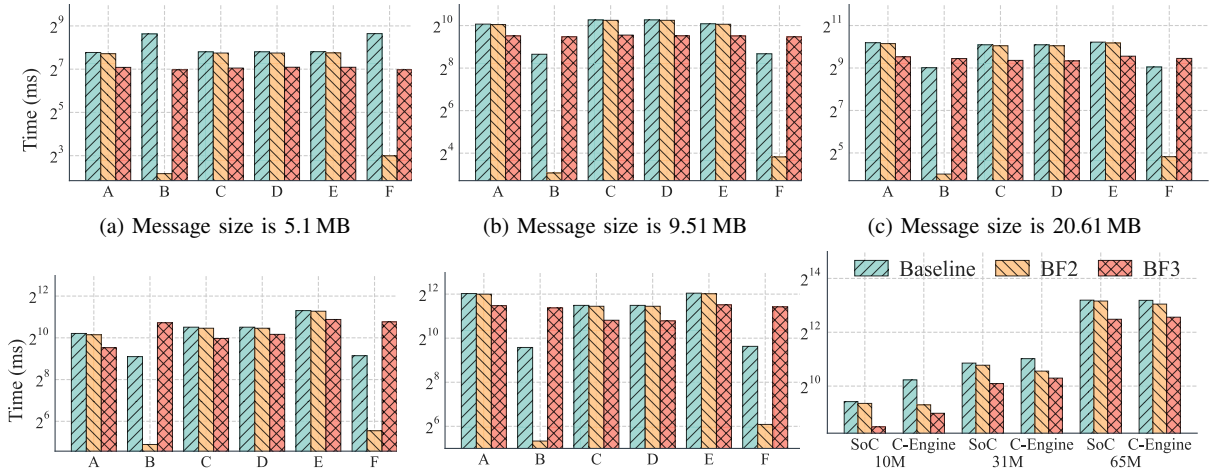
E. Evaluation on MPI Collective Communication – Broadcast

In this section, we assess the enhancement PEDAL offers for MPI collective communication, specifically focusing on the Broadcast operation, while incorporating various compression methods—both lossy and lossless.

Figure 11 presents a comparative analysis of MPI Broadcast time over four nodes, considering diverse message sizes: 5.1 MB (small), 20.6 MB (medium), and 48.8 MB (large). The evaluations are based on whether compression is executed on the SoC or the C-Engine of BlueField-2 and BlueField-3. As a point of reference, similar to Section V-D, our baseline configuration operates on BlueField-2 and integrates compression within the MPI Broadcast. However, this design is hindered by additional overheads—specifically, repeated memory allocations and, if engaged, engine initialization. There are a few critical observations that emerge: utilizing the C-Engine of BlueField-2, there is a substantial reduction in broadcast time—achieving a speedup of up to 68× over the baseline across various compression designs. In the context of BlueField-3, its SoC contributes to an average decrease in MPI Broadcast time by about 49% across the tested message sizes. Conversely, BlueField-3's C-Engine, tailored exclusively for decompression and lacks compression operations, defaults to leveraging its SoC. As a result, compression designs like DEFLATE and zlib, this setup not only fail to surpass the performance of BlueField-2's C-Engine but occasionally even register a slight increase in latency compared to the baseline.

Insights From PEDAL-Optimized MPI Collective Communication

BF2's C-Engine: Offers a speedup of up to 68× over the baseline across compression designs.
BF3's SoC: Reduces MPI Broadcast time by approximately 49% across varied message sizes.
BF3's C-Engine Limitation: Exclusively for decompression operation, sometimes resulting in latency higher than the baseline for specific designs like DEFLATE and zlib.



(a) Message size is 5.1 MB (b) Message size is 9.51 MB (c) Message size is 20.61 MB

(d) Message size is 30 MB (e) Message size is 48.85 MB (f) Comm. with SZ3 lossy compressor

Fig. 10: Comparison of point-to-point communication latency employing lossless (a)-(e) and lossy (f) compression mechanisms as evaluated by the OSU Micro-Benchmarks. In plots (a)-(e), labels A-F correspond to distinct compression techniques, enumerated as: "SoC_DEFLATE", "C-Engine_DEFLATE", "SoC_LZ4", "C-Engine_LZ4", "SoC_zlib", and "C-Engine_zlib". Note that the y-axes are in log scales.

VI. Discussion

Drawing from our findings, we highlight several observations we eagerly share with the broader community. These insights are intended to contribute constructively to the ongoing research and developments in DPU-based data compression.

Sharing experiences with the DPU community: Considering the performance benefits brought by DPU's C-Engine, we recommend that the DPU community consider reintegrating both compression and decompression operations into the C-Engine in their future products. Additionally, there is a strong suggestion for the DPU community to explore expanding compression algorithms or providing programmability that allows developers to customize these C-Engines more extensively.

Sharing experience with the MPI community: Consider exploring alternative deployment scenarios, such as MPI on the host while offloading data compression to the DPU. It is crucial to assess the overhead associated with data movement between the host and DPU, emphasizing the need for a balanced design between computation and communication efficiency. Though this scenario deviates from our paper's primary focus, evaluating computation and communication overlaps, along with pipeline designs, can help alleviate potential performance bottlenecks, particularly in data movement latency.

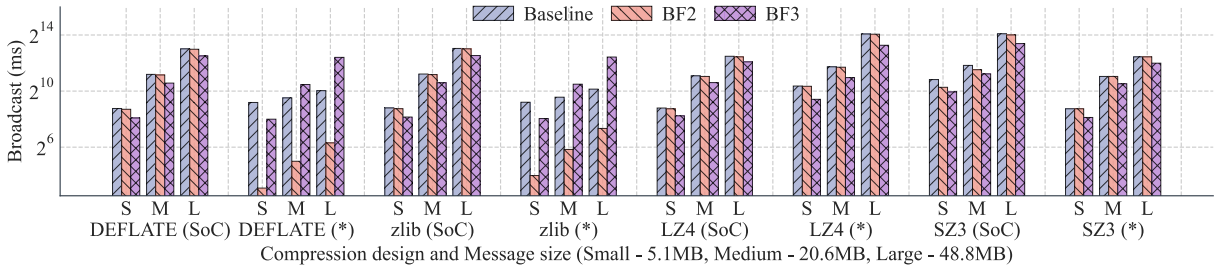
Furthermore, there is an opportunity for the MPI community to advance by developing enhanced compression-extensible infrastructures. Our co-design approach effectively integrates with MPICH, but further investigation into strategies for seamless compression task integration within the MPI framework could unlock new levels of efficiency and performance.

Sharing experience with PEDAL users: Our current methodology centers on improving data compression and decompression efficiency in evolving DPU architectures, seamlessly integrating with the MPICH library under the MPI standard. This ensures the MPI interface remains unchanged, promoting widespread adoption by MPI-based scientific applications. Additionally, the standalone PEDAL library is readily accessible to these applications, offering user-friendly APIs for added convenience. Prospective PEDAL users have the option to either utilize the PEDAL-optimized MPI library or directly program with PEDAL for designing their data compression and decompression pipelines.

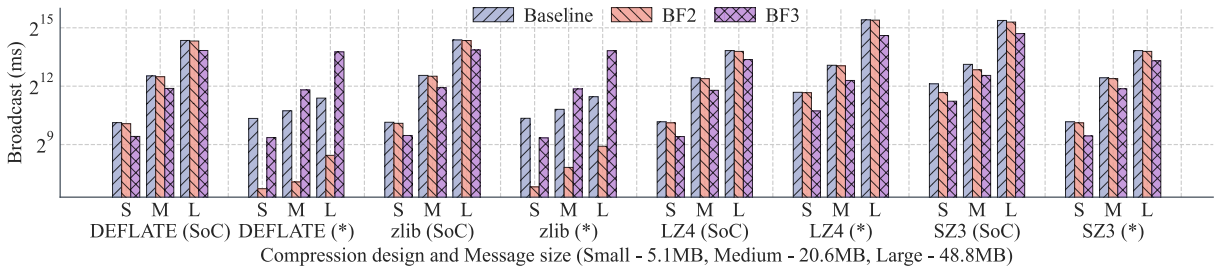
Sharing experience with the data compression community: Leveraging our experience optimizing zlib and SZ3 with SoC and C-Engine synergy, the data compression community can explore more advanced DPU-based compression designs. For instance, using the C-Engine for compression and SoC for metadata generation offers a promising path to enhance performance and inform more efficient compression and decompression schemes.

VII. Related Work

Prior works [12]–[15] on compression and decompression methods focus on reducing data volume while storing scientific datasets. SZ3 [12] uses a modular compression framework on CPUs that allows composing different error-bounded lossy compressors and tailoring various stages of the compression pipeline. SZx [13] achieves high-speed compression on both CPUs and GPUs with error bounds by suppressing expensive arithmetic operations while compressing data. cuSZ [14] and WaveSZ [15] utilize GPUs and field programmable gate arrays (FPGAs), respectively, to parallelize an error-bounded lossy compression framework, SZ [10], [11]. Also, CEAZ [16] proposes a lossy compressor for scientific data on FPGAs-



(a) Four nodes, ppn1



(b) Four nodes, ppn8

Fig. 11: MPI Broadcast with four nodes (* measured on C-Engine). Note that the y-axes are in log scales.

based SmartNIC. Runway [46] deploys SZ3 lossy compression on BF2 and evaluates on the SDRBench benchmark. Our work differs from others as we systematically analyze and accelerate both lossy and lossless compression schemes on BF2 & BF3 DPUs with our proposed uniform library and show its effectiveness for communication-intensive workloads.

On the other hand, some studies evaluated the benefit of compression techniques for reducing the load on communication fabrics. Zhou et al. [47], [48] use GPU-based compression to reduce inter-node data movement bottleneck for HPC applications. cMPI library investigates on-the-fly data compression using CPUs to reduce the communication time of individual messages and to improve the overall bandwidth of large-scale systems [49]. Similarly, CoMPI [50] performs runtime compression of MPI messages via CPUs and reduces the execution time of MPI benchmarks and HPC applications.

There are also studies [51]–[54] analyzing performance attributes of SmartNICs, especially NVIDIA BlueField DPUs. These works focus on DPU SoC core characterizations but our work focus on compression and decompression on SoC core and acceleration. To the best of our knowledge, we are the first to show that DPUs, which are gaining popularity in data center and HPC environments, can be efficiently leveraged to perform data compression and decompression.

VIII. Conclusion and Future Work

Our research marks a milestone by exploring and harnessing NVIDIA BlueField DPUs to accelerate lossy and lossless compression design tailored for communication-oriented workloads in the HPC domain. This pioneering approach has led to remarkable advancements, reducing compression time by up to 101× and slashing data communication latency by as much as 88× through efficient and effective message size reduction. Our study sheds light on the challenges inherent in implementing compression techniques on BlueField-2 and BlueField-3 DPUs, predominantly rooted in hardware constraints. In response to these challenges, we meticulously architected a harmonized co-design approach, ensuring it aligns with the specificities of communication-centric scenarios. Our paper not only identifies these challenges but also presents a unified methodology meticulously crafted to expedite compression and decompression processes on BlueField-2 and BlueField-3 DPUs, specifically tailored for data communication scenarios.

Our future endeavors will further optimize PEDAL along with research directions outlined in Section VI to enhance data compression performance and expand PEDAL's capabilities to support a broader range of applications.

Acknowledgment

We extend our heartfelt gratitude to our anonymous reviewers for their invaluable feedback on the paper. We thank our lab mates, with special appreciation to Liuyao Dai and Hao Qi for their valuable input. We gratefully acknowledge the computing resources provided on Thor, an HPC cluster operated by the HPC Advisory Council, and the generous hardware donation from NVIDIA for the BlueField resources. Part of this research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy (DOE) Office of Science and the National Nuclear Security Administration, and by the U.S. Department of Energy, Office of Science, under Contract DE-AC02-06CH11357. This work was supported in part by NSF research grants OAC #2321123 and #2340982, a subaward granted by Argonne National Laboratory, and a DOE research grant DE-SC0024207.

REFERENCES

[1] H. Xu, C.-Y. Ho, A. M. Abdelmoniem, A. Dutta, E. H. Bergou, K. Karatsenidis, M. Canini, and P. Kalnis, "GRACE: A Compressed Communication Framework for Distributed Machine Learning," in *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*, 2021, pp. 561–572.

[2] Y. Lin, S. Han, H. Mao, Y. Wang, and W. J. Dally, "Deep Gradient Compression: Reducing the Communication Bandwidth for Distributed Training," 2020.

[3] H. Lim, D. G. Andersen, and M. Kaminsky, "3LC: Lightweight and Effective Traffic Compression for Distributed Machine Learning," 2018.

[4] K. Mishchenko, E. Gorbunov, M. Takáč, and P. Richtárik, "Distributed Learning with Compressed Gradient Differences," 2019.

[5] C.-F. Lee, S. W. Changchien, W.-T. Wang, and J.-J. Shen, "A Data Mining Approach to Database Compression," *Information Systems Frontiers*, vol. 8, pp. 147–161, 2006.

[6] S. M. Darwish, "Improving Semantic Compression Specification in Large Relational Database," *IET Software*, vol. 10, no. 4, pp. 108–115, 2016.

[7] M. Hosseini, "A Survey of Data Compression Algorithms and Their Applications," *Network Systems Laboratory, School of Computing Science, Simon Fraser University, BC, Canada*, 2012.

[8] T. Srisooksai, K. Keamarungsi, P. Lamsrichan, and K. Araki, "Practical Data Compression in Wireless Sensor Networks: A Survey," *Journal of Network and Computer Applications*, vol. 35, no. 1, pp. 37–59, 2012, collaborative Computing and Applications. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804511000555

[9] Y. Li, H. Qi, G. Lu, F. Jin, Y. Guo, and X. Lu, "Understanding hot interconnects with an extensive benchmark survey," *BenchCouncil Transactions on Benchmarks, Standards and Evaluations*, vol. 2, no. 3, p. 100074, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2772485922000618

[10] D. Tao, S. Di, Z. Chen, and F. Cappello, "Significantly Improving Lossy Compression for Scientific Data Sets Based on Multidimensional Prediction and Error-Controlled Quantization," in *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2017, pp. 1129–1139.

[11] S. Di and F. Cappello, "Fast Error-Bounded Lossy HPC Data Compression with SZ," in *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2016, pp. 730–739.

[12] X. Liang, K. Zhao, S. Di, S. Li, R. Underwood, A. M. Gok, J. Tian, J. Deng, J. C. Calhoun, D. Tao, Z. Chen, and F. Cappello, "SZ3: A Modular Framework for Composing Prediction-Based Error-Bounded Lossy Compressors," *IEEE Transactions on Big Data*, vol. 9, no. 2, pp. 485–498, 2023.

[13] X. Yu, S. Di, K. Zhao, J. Tian, D. Tao, X. Liang, and F. Cappello, "Ultrafast Error-Bounded Lossy Compression for Scientific Datasets," in *Proceedings of the 31st International Symposium on High-Performance Parallel and Distributed Computing*, ser. HPDC '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 159–171. [Online]. Available: https://doi.org/10.1145/3502181.3531473

[14] J. Tian, S. Di, K. Zhao, C. Rivera, M. H. Fulp, R. Underwood, S. Jin, X. Liang, J. Calhoun, D. Tao, and F. Cappello, "cuSZ: An Efficient GPU-Based Error-Bounded Lossy Compression Framework for Scientific Data," in *Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 3–15. [Online]. Available: https://doi.org/10.1145/3410463.3414624

[15] J. Tian, S. Di, C. Zhang, X. Liang, S. Jin, D. Cheng, D. Tao, and F. Cappello, "WaveSZ: A Hardware-Algorithm Co-Design of Efficient Lossy Compression for Scientific Data," in *Proceedings of the 25th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 74–88. [Online]. Available: https://doi.org/10.1145/3332466.3374525

[16] C. Zhang, S. Jin, T. Geng, J. Tian, A. Li, and D. Tao, "CEAZ: Accelerating Parallel I/O via Hardware-Algorithm Co-Designed Adaptive Lossy Compression," in *Proceedings of the 36th ACM International Conference on Supercomputing*, ser. ICS '22. New York, NY, USA: Association for Computing Machinery, 2022. [Online]. Available: https://doi.org/10.1145/3524059.3532362

[17] NVIDIA, "NVIDIA BLUEFIELD-2 DPU," https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/documents/datasheet-nvidia-bluefield-2-dpu.pdf, 2023 (Accessed on 2023-04-02).

[18] ——, "NVIDIA BLUEFIELD-3 DPU," https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/documents/datasheet-nvidia-bluefield-3-dpu.pdf, 2023 (Accessed on 2023-04-05).

[19] Y. Li, A. Kashyap, Y. Guo, and X. Lu, "Characterizing Lossy and Lossless Compression on Emerging BlueField DPU Architectures," in *Proceedings of the 30th IEEE Hot Interconnects Symposium*, ser. HotI '23. IEEE Computer Society, August 2023.

[20] ——, "Compression Analysis for BlueField-2/3 DPUs: Lossy and Lossless Perspectives," *IEEE Micro*, no. 01, pp. 1–9, dec 5555.

[21] NVIDIA, "NVIDIA DOCA Software Framework," https://developer.nvidia.com/networking/doca, 2023.

[22] P. Deutsch, "Rfc1951: Deflate Compressed Data Format Specification Version 1.3," 1996.

[23] J.-l. Gailly and M. Adler, "zlib," https://zlib.net/, 2022.

[24] Z. Zhang, C. Chang, H. Lin, Y. Wang, R. Arora, and X. Jin, "Is Network the Bottleneck of Distributed Training?" in *Proceedings of the Workshop on Network Meets AI & ML*, ser. NetAI '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 8–13. [Online]. Available: https://doi.org/10.1145/3405671.3405810

[25] A. Sapio, M. Canini, C.-Y. Ho, J. Nelson, P. Kalnis, C. Kim, A. Krishnamurthy, M. Moshref, D. Ports, and P. Richtarik, "Scaling Distributed Machine Learning with In-Network Aggregation," in *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. USENIX Association, Apr. 2021, pp. 785–808. [Online]. Available: https://www.usenix.org/conference/nsdi21/presentation/sapio

[26] C. Lao, Y. Le, K. Mahajan, Y. Chen, W. Wu, A. Akella, and M. Swift, "ATP: In-network Aggregation for Multi-tenant Learning," in *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. USENIX Association, Apr. 2021, pp. 741–761. [Online]. Available: https://www.usenix.org/conference/nsdi21/presentation/lao

[27] X. Lu, M. W. U. Rahman, N. Islam, D. Shankar, and D. K. Panda, "Accelerating Spark with RDMA for Big Data Processing: Early Experiences," in *2014 IEEE 22nd Annual Symposium on High-Performance Interconnects*, 2014, pp. 9–16.

[28] X. Lu, F. Liang, B. Wang, L. Zha, and Z. Xu, "DataMPI: Extending MPI to Hadoop-Like Big Data Computing," in *2014 IEEE 28th International Parallel and Distributed Processing Symposium*, 2014, pp. 829–838.

[29] X. Zhang, B. Liu, Z. Gou, J. Shi, and X. Zhao, "DCache: A Distributed Cache Mechanism for HDFS based on RDMA," in *2020 IEEE 22nd International Conference on High Performance Computing and Communications; IEEE 18th International Conference on Smart City; IEEE 6th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, 2020, pp. 283–291.

[30] N. S. Islam, X. Lu, M. Wasi-ur Rahman, D. Shankar, and D. K. Panda, "Triple-H: A Hybrid Approach to Accelerate HDFS on HPC Clusters with Heterogeneous Storage Architecture," in *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE, 2015, pp. 101–110.

[31] "LZ4," https://github.com/lz4/lz4, 2023.

[32] "MPICH," https://www.mpich.org/, 2023.

[33] NVIDIA, "Power the Next Wave of Applications with NVIDIA BlueField-3 DPUs," https://developer.nvidia.com/blog/power-the-next-wave-of-applications-with-nvidia-bluefield-3-dpus/#:~:text=BlueField%2D3%20has%202x%20the,the%20NVIDIA%20DOCA%20software%20framework, 2021, accessed: 2023-02-21.

[34] "RDMA Stack Support on Host and Arm System," 2023. [Online]. Available: https://docs.nvidia.com/networking/display/bluefielddpuosv390/rdma+stack+support+on+host+and+arm+system#src-89159667_RDMAStackSupportonHostandArmSystem-EmbeddedCPUMode

[35] J. Ziv and A. Lempel, "A Universal Algorithm for Sequential Data Compression," *IEEE Transactions on Information Theory*, vol. 23, no. 3, pp. 337–343, 1977.

[36] D. A. Huffman, "A Method for the Construction of Minimum-Redundancy Codes," *Proceedings of the IRE*, vol. 40, no. 9, pp. 1098–1101, 1952.

[37] "Unified Communication X," 2015. [Online]. Available: https://openucx.org/

[38] "Libfabric OpenFabrics," 2019. [Online]. Available: https://ofiwg.github.io/libfabric/

[39] H. Zhou, K. Raffenetti, W. Bland, and Y. Guo, "Generating Bindings in MPICH," 2024.
[40] "SDRBench," 2024. [Online]. Available: https://sdrbench.github.io
[41] K. Zhao, S. Di, X. Lian, S. Li, D. Tao, J. Bessac, Z. Chen, and F. Cappello, "SDRBench: Scientific Data Reduction Benchmark for Lossy Compressors," in *2020 IEEE International Conference on Big Data (Big Data)*. Los Alamitos, CA, USA: IEEE Computer Society, dec 2020, pp. 2716–2724. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/BigData50022.2020.9378449
[42] "Silesia Compression Corpus," 2024. [Online]. Available: https://sun.aei.polsl.pl/~sdeor/index.php?page=silesia
[43] M. Burtscher and P. Ratanaworabhan, "FPC: A High-Speed Compressor for Double-Precision Floating-Point Data," *IEEE Transactions on Computers*, vol. 58, no. 1, pp. 18–31, 2009.
[44] "HPC Advisory Council," https://hpcadvisorycouncil.atlassian.net/wiki/spaces/HPCWORKS/overview, 2023.
[45] "OSU Micro-Benchmarks," https://mvapich.cse.ohio-state.edu/benchmarks/, 2023.
[46] J. Ravi, S. Byna, and M. Becchi, "Runway: In-transit Data Compression on Heterogeneous HPC Systems," in *2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, 2023, pp. 229–239.
[47] Q. Zhou, C. Chu, N. S. Kumar, P. Kousha, S. M. Ghazimirsaeed, H. Subramoni, and D. K. Panda, "Designing High-Performance MPI Libraries with On-the-fly Compression for Modern GPU Clusters," in *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2021, pp. 444–453.
[48] Q. Zhou, P. Kousha, Q. Anthony, K. Shafie Khorassani, A. Shafi, H. Subramoni, and D. K. Panda, "Accelerating MPI All-to-All Communication with Online Compression on Modern GPU Clusters," in *High Performance Computing*, A.-L. Varbanescu, A. Bhatele, P. Luszczek, and B. Marc, Eds. Cham: Springer International Publishing, 2022, pp. 3–25.
[49] J. Ke, M. Burtscher, and E. Speight, "Runtime Compression of MPI Messanes to Improve the Performance and Scalability of Parallel Applications," in *SC '04: Proceedings of the 2004 ACM/IEEE Conference on Supercomputing*, 2004, pp. 59–59.
[50] R. Filgueira, D. E. Singh, A. Calderón, and J. Carretero, "CoMPI: Enhancing MPI Based Applications Performance and Scalability Using Run-Time Compression," in *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, M. Ropo, J. Westerholm, and J. Dongarra, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 207–218.
[51] J. Liu, C. Maltzahn, C. D. Ulmer, and M. L. Curry, "Performance Characteristics of the BlueField-2 SmartNIC," 5 2021. [Online]. Available: https://www.osti.gov/biblio/1783736
[52] X. Wei, R. Cheng, Y. Yang, R. Chen, and H. Chen, "Characterizing Off-path SmartNIC for Accelerating Distributed Systems," in *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*. Boston, MA: USENIX Association, Jul. 2023, pp. 987–1004. [Online]. Available: https://www.usenix.org/conference/osdi23/presentation/wei-smartnic
[53] B. Michalowicz, K. K. Suresh, H. Subramoni, D. K. D. Panda, and S. Poole, "Battle of the BlueFields: An In-Depth Comparison of the BlueField-2 and BlueField-3 SmartNICs," in *2023 IEEE Symposium on High-Performance Interconnects (HOTI)*, 2023, pp. 41–48.
[54] X. Chen, J. Zhang, T. Fu, Y. Shen, S. Ma, K. Qian, L. Zhu, C. Shi, M. Liu, and Z. Wang, "Demystifying Datapath Accelerator Enhanced Off-path SmartNIC," *arXiv preprint arXiv:2402.03041*, 2024.