Streamlining HDF5's AI Workloads Benchmarking

Dlyaver Djebarov
RWTH Aachen University
Aachen, Germany
dlyaver.djebarov@rwth-aachen.de

Radita Liem
RWTH Aachen University
Aachen, Germany
liem@itc.rwth-aachen.de

Sarah Neuwirth
Johannes Gutenberg University Mainz
Mainz, Germany
neuwirth@uni-mainz.de

Jean Luca Bez
Lawrence Berkeley National Laboratory
Berkeley, USA
jlbez@lbl.gov

Suren Byna
The Ohio State University
Columbus, USA
byna.1@osu.edu

***Abstract*—Rapid adoption of artificial intelligence (AI) in scientific computing requires new tools to evaluate I/O performance effectively. HDF5 is one of the data formats frequently used not only in HPC but also in modern AI applications. However, existing benchmarks are insufficient to address the current challenges posed by AI workloads. This paper introduces an extension to the existing HDF5 benchmark, called h5bench, by incorporating the workload characteristics from the MLPerf Storage - DLIO benchmark. This extension allows users to test AI workloads together with traditional HPC benchmarks in the same context without the complexities of installing various machine learning libraries. Our experimental analysis demonstrates that the extension can replicate the existing I/O patterns with easily customizable configurations to perform various scaling tests.**

***Index Terms*—Benchmarking, HDF5, AI Workloads, DLIO**

I. Introduction

Artificial intelligence (AI) has been rapidly adopted in various scientific disciplines, as described by Xu et al. [1]. This rapid adoption of AI increases the need for I/O systems that can meet the demands of these workloads. Some of the strategies are to develop new specialized systems to accommodate this emerging AI workload [2], [3] as the I/O performance bottleneck problem becomes more pronounced. Despite the critical impact of I/O performance in AI applications, there are a limited number of benchmarks that specifically address this problem. In previous studies [4], [5]. the Deep Learning I/O (DLIO) benchmark [6] has been identified as the leading I/O benchmark for AI workloads, which is also part of the MLPerf Storage benchmark suite [7].

Aside from the lack of options for benchmarking I/O in AI workloads, we have identified several other issues in this area. First, most HPC systems are not used exclusively for AI workloads, but are shared with several other traditional HPC applications. Setting up a system that can accommodate both workloads is an essential task for the stakeholders of any HPC center. It is important to put the result of the AI-focused benchmark in the same context as the result from the benchmark for more traditional workloads to obtain the right conclusion for decision making when tuning the systems. Second, existing AI benchmarks require the installation of machine learning libraries, which can affect the metrics being evaluated. Setting up the appropriate environment for running AI experiments is also one of the productivity challenges that developers and performance analysts face when evaluating AI workloads [8]. This complexity needs to be navigated to produce a holistic view for tuning and managing a cluster.

With these two problems in mind, we propose an extension to the existing I/O benchmark suite called *h5bench* [9] to enable benchmarking of AI workload. We specifically target the Hierarchical Data Format version 5 (HDF5) [10], as HDF5 is one of the data formats commonly used not only in HPC applications [11] but also in various AI applications. h5bench itself already has existing configurations for testing various workload scenarios for traditional HPC applications, and this structure can be extended to accommodate the scenarios for AI workloads. In our prototype, we are looking at DLIO workloads to incorporate the specific configuration requirements for AI applications into our h5bench extension.

The scientific contribution of our extension is to enable holistic I/O benchmarking that covers the traditional and AI use cases, and in our evaluation study, we pinpoint the specific impact of the Python library on the I/O operations by isolating the HDF5 operations in our h5bench extension.

Our paper is structured as follows. In Section I we present the current situation and challenges in benchmarking I/O components in AI workloads. In Section II we list existing benchmarks targeting I/O and AI workloads in general. Section III contains a brief background on h5bench, followed by the scope of AI workloads we have integrated in Section IV. Our implementation steps are explained in Section V. We explain our experimental setup in Section VI, and the results can be found in Section VII. In Section VIII we discuss the limitations of our current work and future work.

II. Related Work

When it comes to the most relevant tool for I/O benchmarking today, IOR [12] is the leading parallel I/O benchmark for evaluating the bandwidth performance of parallel filesystems. IOR supports different file access patterns (i.e., read and write, single shared file, and file-per-process) and multiple I/O library interfaces (i.e., POSIX, MPI-IO, and HDF5). IOR

also offers several interfaces for distributed computing such as S3 and HDFS. In the IOR code repository, there is also MDTest [13], which is used to evaluate the metadata performance of POSIX-compliant parallel filesystems. MDTest measures the performance of file, directory, and directory hierarchy operations, including creation, stat, and removal. Users can create extensive combinations of I/O patterns with IOR and MDTest, as seen in the IO500 [14] benchmark suite, making this benchmark very powerful, but also overwhelming for users. Besides h5bench [9] that is used in this work, there are multiple other existing HDF5 benchmarks that are based on the application patterns as already listed in the work of Bez et al.[9], such as MACSio [15], Parallel I/O Kernel Suite [16], FLASH-IO [17], ChomboIO [18], AMReX [19].

When it comes to benchmarking AI and Machine Learning (ML) applications that focus on the performance in HPC systems, MLPerf [20] is one of the most recognized benchmarks in the field. MLPerf is organized by the MLCommons working group [21] and offers a variety of applications that use machine learning and deep learning in HPC environments. It focuses on several aspects, such as inference, training and storage. DLIO [6] is part of the MLPerf Storage [7]. In the broader context of ML benchmarking, there are several other benchmarks listed by Thiyagalingam et al. [22], such as Deep500 [23], RLBench [24], DAWNBench [25] and AIBench [26].

Several survey and analysis works are also trying to look into I/O patterns and characterizations [4], [27]–[29]. Based on the survey paper by Bez et al. [4], we examine DLIO as the basis for our work, as it is currently the most widely known I/O benchmark for AI workloads.

III. H5BENCH

h5bench [9] is the HDF5 benchmark that we use in this work. It is a benchmark that is created to provide a set of HDF5 I/O kernels that can represent applications using the HDF5 API while optimizing their I/O performance by leveraging the new features such as HDF5 Virtual Object Layer (VOL) connector. In Figure 1 an overview of h5bench is shown, in which the benchmark can execute a single kernel or several patterns like a workflow. The configuration can be done via a single JSON file.

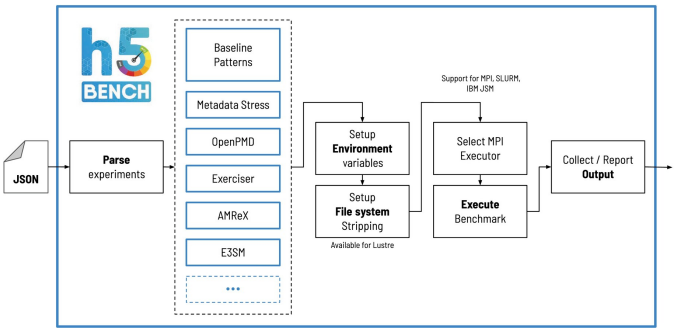

Fig. 1: The overview of h5bench suite architecture. Figure is taken from the work of Bez et al. [9].

There are five main components within the JSON input: *mpi*, *vol*, *file-system*, *directory*, and *benchmarks*. *mpi* configures the MPI launcher (`mpirun`, `mpiexec`, `srun`, `qsub`, and `jsrun`) and the number of processes information. The features of the VOL connectors can be configured in the *vol* section. The *file-system* handles the striping configuration for Lustre. Finally, *benchmarks* covers the detailed properties of the patterns, such as execution order, input, chunking, and so on. Our h5bench extension will expand on the *benchmarks* component.

VOL connectors are one of the HDF5 features highlighted in h5bench. Multiple VOL configurations can be used to create distinct I/O patterns, such as asynchronous I/O VOL connector, which enables asynchronous I/O for HDF5 operations using background threads. Another example is the Cache VOL connector, which enables data caching on the node-local storage and moves the data asynchronously between the node-local storage and the parallel file systems, hiding the I/O overhead. Not all benchmarks inside h5bench support all VOL connectors, and our current prototype for the AI workload extension does not yet use a VOL connector.

We chose h5bench because it provides the completeness of the access pattern as discussed by Bez et al. [4]. The extension of h5bench is a good starting point for a holistic view of the I/O performance of the cluster. In addition, HDF5 is frequently used in traditional HPC and AI workloads [30].

IV. AI WORKLOADS

In the context of our literature review, we decided to investigate the DLIO benchmark [6] in more detail due to its importance in the current research landscape. Similarly to the concept of the h5bench, the DLIO benchmark was developed to emulate the I/O patterns and behaviors typical of deep learning applications. This benchmark suite can be configured to accommodate different data loaders, data formats, dataset organizations, and training configuration parameters. The configuration is done using a YAML file. The configuration of DLIO consists of six main components:

- ***workflow*** is where users specify the stage of the machine learning workflow that will be emulated (data generation, training, evaluation). In addition, it has checkpointing and profiling options.
- ***dataset*** outlines the data setup for the specified stage in *workflow*. It provides details on record length, variations in record length, the number of files used for training and evaluation, the number of samples per file, HDF5 chunking settings, and other relevant parameters.
- ***reader*** covers the batch size, number of read threads needed to load the data, transfer size, shuffle size, and preprocess time (using sleep).
- ***train*** defines epochs, training steps, and computation time, including their variation, are configured here.
- ***evaluation*** covers the time for the evaluation and the epochs between evaluations.
- ***checkpoint*** handles the checkpoint configuration.

These configurations, except for checkpointing, are integrated into our h5bench extension and will be discussed in more detail in Section VII.

Similar to h5bench, where users can create customized configurations for DLIO or use existing ones, the DLIO code repository[1] provides several ready-to-use configurations based on real-world deep learning applications, such as CosmoFlow, BERT, and UNET3D. In Table I, we have compiled a selection of notable workloads from the DLIO examples to provide an overview of the configurations used in our experiments and highlight their differences from those of other AI applications.

TABLE I: Configurations of deep learning kernels in DLIO.

Benchmark	CosmoFlow	BERT	UNET3D
backend	tensorflow	tensorflow	pytorch
record-length	2828486	2500	146600628
num-files-train	524288	500	168
num-samples-per-file	1	313532	1
batch-size	1	48	7
read-threads	4	1	4
epochs	5	1	5

In Table I, we include only the parameters that significantly impact the runtime of AI applications and have distinct values across different applications. DLIO summarizes the key components that differentiate AI workloads, primarily based on the configuration elements in *dataset* and *reader*.

As DLIO aims to emulate the full impact of Python libraries, running DLIO requires the usual settings for running real-world deep learning applications, including the installation of machine learning library dependencies such as Tensorflow [31] and PyTorch [32], as we saw in the backend specified in Table I. In addition to HDF5, which we will use in our extension, DLIO also supports other common AI data formats such as TFRecord [33] and npz [34].

Although the overall machine learning environment and settings can impact I/O performance, it is also crucial for cluster stakeholders to gain a holistic view of various HDF5 workloads. Effective cluster tuning requires balancing multiple workload types beyond just AI. This broader perspective can serve as a baseline for quantifying the impact of AI backends and libraries. We discuss this topic further in Section VII-A.

V. H5BENCH EXTENSION

Our effort to develop an extension that can represent AI workloads begins with an examination of the workloads described in Section IV. We analyzed DLIO a nd summarized its existing configurations[2] to identify its general structure and key components relevant to our work. Following this analysis, our approach is primarily divided three steps: into pattern extraction, implementation, and testing.

[1]https://github.com/argonne-lcf/dlio_benchmark/tree/main/dlio_benchmark/configs/workload

[2]https://dlio-benchmark.readthedocs.io/en/latest/config.html

A. Pattern Extraction

To extract patterns from existing HDF5 applications, we utilize log VFD. Virtual File Drivers (VFD) is an API layer in HDF5 that creates mappings between HDF5 address space and storage. Log VFD is part of a feature designed to capture metrics and access activity for HDF5 files, enabling the extraction of access patterns. It provides detailed information about file operations, memory allocations, and dataset reads/writes, which are crucial for analyzing the I/O patterns of AI workloads. Log VFD can be activated by setting the environment variable `HDF5_DRIVER` to `log`. The generated log is directed to `stderr` stream. An example of this output can be seen below:

```
16-95 (80 bytes) (H5FD_MEM_SUPER) Read
```

The first two numbers are the range of bytes that were accessed or allocated in the HDF5 file, with the number of bytes in this range given in brackets. The next bracket contains the type of memory being accessed. The last piece of information is the type of operation performed on this file fragment. There is also a way to modify the file access property in the source code using the function `H5Pset_fapl_log()`.

The following is the log collected when reading the first sample of an HDF5 file in DLIO. Samples have already been mentioned in Section IV and represent the unit of training data. Typically, an HDF5 file contains several samples. In the following example, one file contained four samples:

```
0-8           (8 bytes)           (H5FD_MEM_SUPER) Allocated
0-7           (8 bytes)           (H5FD_MEM_SUPER) Read
8-16          (8 bytes)           (H5FD_MEM_SUPER) Allocated
0-15          (16 bytes)          (H5FD_MEM_SUPER) Read
16-96         (80 bytes)          (H5FD_MEM_SUPER) Allocated
16-95         (80 bytes)          (H5FD_MEM_SUPER) Read
96-268437536  (268437440 bytes)   (H5FD_MEM_DEFAULT) Allocated
96-607        (512 bytes)         (H5FD_MEM_OHDR) Read
680-1191      (512 bytes)         (H5FD_MEM_LHEAP)
136-679       (544 bytes)         (H5FD_MEM_BTREE) Read
1072-1399     (328 bytes)         (H5FD_MEM_BTREE) Read
800-1311      (512 bytes)         (H5FD_MEM_OHDR) Read
2048-67110911 (67108864 bytes)    (H5FD_MEM_DRAW) Read
```

The log above shows that the small `H5FD_MEM_SUPER memory` is used intensively at the beginning to manage the structure, e.g. to identify the file as an HDF5 file and to determine the address of the first block of the file in order to find other data structures. This is followed by data allocation in the memory with the size of 268437440 bytes ($\approx$ 256 MiB), which corresponds to the size of the data contained in the file with little overhead. The memory type `H5FD_MEM_DEFAULT` indicates that the memory type has not yet been defined. It can also be the datatype that was defined in a more significant allocation that is be sub-allocated by the library.

After reading information from `H5FD_MEM_OHDR` – an object header that provides dataspace, datatype, and other information that the library uses to speed up access to the object's data, `H5FD_MEM_LHEAP` – a local heap is a collection of small pieces of data related to metadata, such as small attribute names and values, and `H5FD_MEM_BTREE` that contains a B-Tree search tree that stores various addresses in a file, the raw content of the HDF5 file is read directly, namely the first sample, whose size is 256 MiB/4 or 64 MiB. When reading the second, third, and fourth samples, the logs differ only in

the last line and contain the intervals $[67110912-134219775]$, $[134219776-201328639]$, and $[201328640-268437503]$.

B. Implementation

The general structure of the implementation follows the DLIO component *workflow*, which consists of data generation, training, and evaluation. From the log collected in the previous steps, we observed that HDF5 usage in DLIO is focused on read operations. Therefore, we follow the current sample as the first step. HDF5 checkpointing operations have not been implemented in our extension, as checkpointing is not part of DLIO's HDF5 operations. While checkpointing is an interesting part of AI workloads, we defer its discussion to future work (Section VIII) and focus solely on implementing components that appear in the HDF5 log VFD. This decision ensures that we can confidently validate our implementation.

We use our previous findings as the basis of our implementation to create the DLIO read pattern (`read_sample`), which provides us with the read time and metadata time:

```
function read_sample(
    file_path,
    sample,
    *metadata_time_out,
    *read_time_out
):
```

This `read_sample` function contains three main sections: (1) initialization for the read and metadata, (2) read operations and finally, (3) collection of time data and release of resources.

```
    /* (1) Initialization */
    offset = [sample, 0, 0]
    count = [1, DIM, DIM]

    t1 = get_current_time_in_microseconds()

    file_id = H5Fopen(file_path, H5F_ACC_RDONLY, FAPL)
    dataset_id = H5Dopen(file_id, RECORDS_DATASET_NAME
     , DAPL)
    filespace = H5Dget_space(dataset_id)
    memspace = H5Screate_simple(3, count, NULL)
    H5Sselect_hyperslab(filespace, H5S_SELECT_SET,
     offset, NULL, count, NULL)

    t2 = get_current_time_in_microseconds()

    data = allocate_memory(DIM * DIM * size_of(uint8_t
     ))

    /* (2) Collecting read information */
    t3 = get_current_time_in_microseconds()
    H5Dread(dataset_id, H5T_STD_U8LE, memspace,
     filespace, DXPL, data)
    t4 = get_current_time_in_microseconds()

    /* (3) Releasing resources */
    free_memory(data)

    t5 = get_current_time_in_microseconds()

    H5Sclose(memspace)
    H5Sclose(filespace)
    H5Dclose(dataset_id)
    H5Fclose(file_id)

    t6 = get_current_time_in_microseconds()
```

The read time and the metadata time collected at the end of the function are important for observing the benchmark scaling. It is defined as follows:

```
/* C3) Collecting time information */
*metadata_time_out = (t2 - t1) + (t6 - t5)
*read_time_out = t4 - t3
```

Our computation time is emulated similarly to DLIO with the function `sleep` with a configurable duration from the parameters of the JSON file. To calculate the I/O time, we use the same method as for the other workloads in h5bench, with the counter starting before the HDF5 function and ending directly after the HDF5 functions. This is to ensure that our reported I/O time and bandwidth remain consistent within h5bench. Users can evaluate the results of traditional HPC workloads in the current h5bench, and the AI workloads in our extension work with the same context.

C. Validation

We implemented several validation strategies to test our extension implementation:

- Our first step is to compare the access patterns produced by the benchmark and the original DLIO code with the same settings. We compared the log VFD produced by our extension with the log VFD produced by DLIO in various settings.
- Then, we conduct a runtime and pattern analysis using DFTracer [35] for DLIO and our extension. We performed an in-depth I/O pattern comparison analysis to check the consistency and identify any discrepancy between our extension and the original DLIO benchmark. We chose DFTracer as our evaluation tool as it was originally developed for DLIO profiling, which is suitable for a fair evaluation of our work.

D. AI Workloads Parameters

As mentioned in Section III, we extend the content of *benchmarks* section of the configuration file with AI workloads-specific parameters. The list of new parameters and their default values are listed in Table II. Note that not all parameters are listed in this table and this table only contains the parameters relevant for AI workloads. The table contains color codes with a white background indicating different stages of the AI application workflow. The rows with yellow backgrounds contain notable parameters that can be found in DLIO configurations to create different patterns. The rows with green color represent HDF5-specific features. Of particular interest are the *chunking* and *chunk-size* parameters, which are HDF5-specific features but can also be found in DLIO, which is why they are highlighted in yellow.

VI. Experimental Setup

The CLAIX-23 [36] supercomputer cluster at RWTH Aachen University was used for our experiments. This cluster consists of Intel Xeon 8468 Sapphire Rapids CPUs with a total of 96 cores per compute node. CLAIX-23 is a fat-tree topology cluster utilizing the InfiniBand network with a unidirectional bandwidth of up to 25 GB/s between nodes. We tested two filesystems: Lustre and BeeOND (BeeGFS On Demand). The specifications are as follows:

TABLE II: h5bench extension's configurations. Rows with yellow backgrounds are parameters found in DLIO configurations to create different patterns. Rows with green color are for HDF5-specific features.

Parameter	Description	Type	Default
generate-data	Enable generation of benchmarking data	bool	$false$
train	Enable model training simulation	bool	$false$
evaluation	Enable model evaluation simulation	bool	$false$
record-length	Record size of a single sample in bytes	int	67108864
num-files-train	The number of files used to train the model	int	32
num-files-eval	The number of files used to evaluate the model	int	8
num-samples-per-file	The number of samples in each file	int	4
chunking	Enable chunking	bool	$false$
chunk-size	Chunk size	int	1024
batch-size	Training batch size	int	7
batch-size-eval	Evaluation batch size	int	2
shuffle	Enable samples shuffle	bool	$false$
preprocess-time	Preprocessing time after reading each sample in seconds	float	0.0
preprocess-time-stdev	Standard deviation in preprocessing time in seconds	float	0.0
epochs	The number of epochs	int	5
total-training-steps	Maximum number of steps per training per epoch	int	-1
computation-time	Computation time after reading each batch in seconds	float	0.323
computation-time-stdev	Standard deviation in computation time in seconds	float	0.0
random-seed	Random seed to be used	int	42
eval-time	Evaluation time after reading each batch in seconds	float	0.323
eval-time-stdev	Standard deviation in evaluation time in seconds	float	0.0
epochs-between-evals	The number of epochs between evaluations	int	1
seed-change-epoch	Enable seed changes every epoch	bool	$false$
read-threads	The number of workers used to read the data	int	4
collective-meta	Enable collective HDF5 metadata operations	bool	$false$
collective-data	Enable collective HDF5 data operations	bool	$false$
subfiling	Enable HDF5 Subfiling Virtual File Driver	bool	$false$

- **Lustre** - RAID0 type. Stripe count 1 for file size less than 1 GB, stripe count 4 for file size between $1-64$ GB, stripe count 16 for file size larger than 64 GB. The stripe size is 16 MB. CLAIX-23's Lustre is a production-level shared storage resource where other users' activities in Lustre can affect our experiments.
- **BeeOND** - RAID0 type with 512K chunk size, four storage targets, and one storage pool. Each node for the BeeOND filesystem has a 1.4 TB SSD. Unlike the Lustre filesystem, BeeOND nodes are used exclusively and are not shared with other users. We can see the impact of this exclusive usage in Figure 5.

We used Intel compilers 2022 with Intel MPI 2021 to build our h5bench extension. Our DLIO experiment used Python 3.10.4, PyTorch 2.3.1, and Tensorflow 2.16.2. Our experiments ran with at least five repetitions to check the variability of the performance numbers. The list of experiments is as follows:

- **Runtime and pattern analysis of the UNET3D workload**: We ran DLIO and the h5bench extension on four nodes (32 processes in total) in the BeeOND environment, as shown in Figure 3, to evaluate our implementation and ensure the consistency of the results.
- **DFTracer analysis**: We ran the UNET3D configuration and also a small case of eight training files with four samples per file to demonstrate the difference and consistency between DLIO and our implementation.
- **Scaling test**: We ran the h5bench extension with 1 to 32 nodes with 24 processes for each node in Lustre and BeeOND with 384 GB of data for training phase and 96 GB for evaluation phase.

VII. Results

A. Implementation Validation

As described in Section V-C, we used several strategies to validate our implementation. The first step was to compare the log VFD outputs to ensure they matched those of DLIO. We verified that the log VFD generated by the h5bench extension is identical to the log VFD produced by DLIO. The procedure for generating these log follows the same method outlined in Section V-A.

Our next step involves conducting a runtime and trace analysis on our h5bench extension in comparison to DLIO runs using the same configurations. In our naive runtime measurement of UNET3D [37] configurations, using DLIO with a Pytorch backend and our extension with identical settings, we observed a consistent runtime difference in our experiment on the BeeOND file system, as shown in Figure 3. This result is expected since our extension is not a one-to-one translation of DLIO. TO enhance user understanding, we provide an in-depth analysis of these differences, ensuring transparency in how our extension models AI workload behaviors.

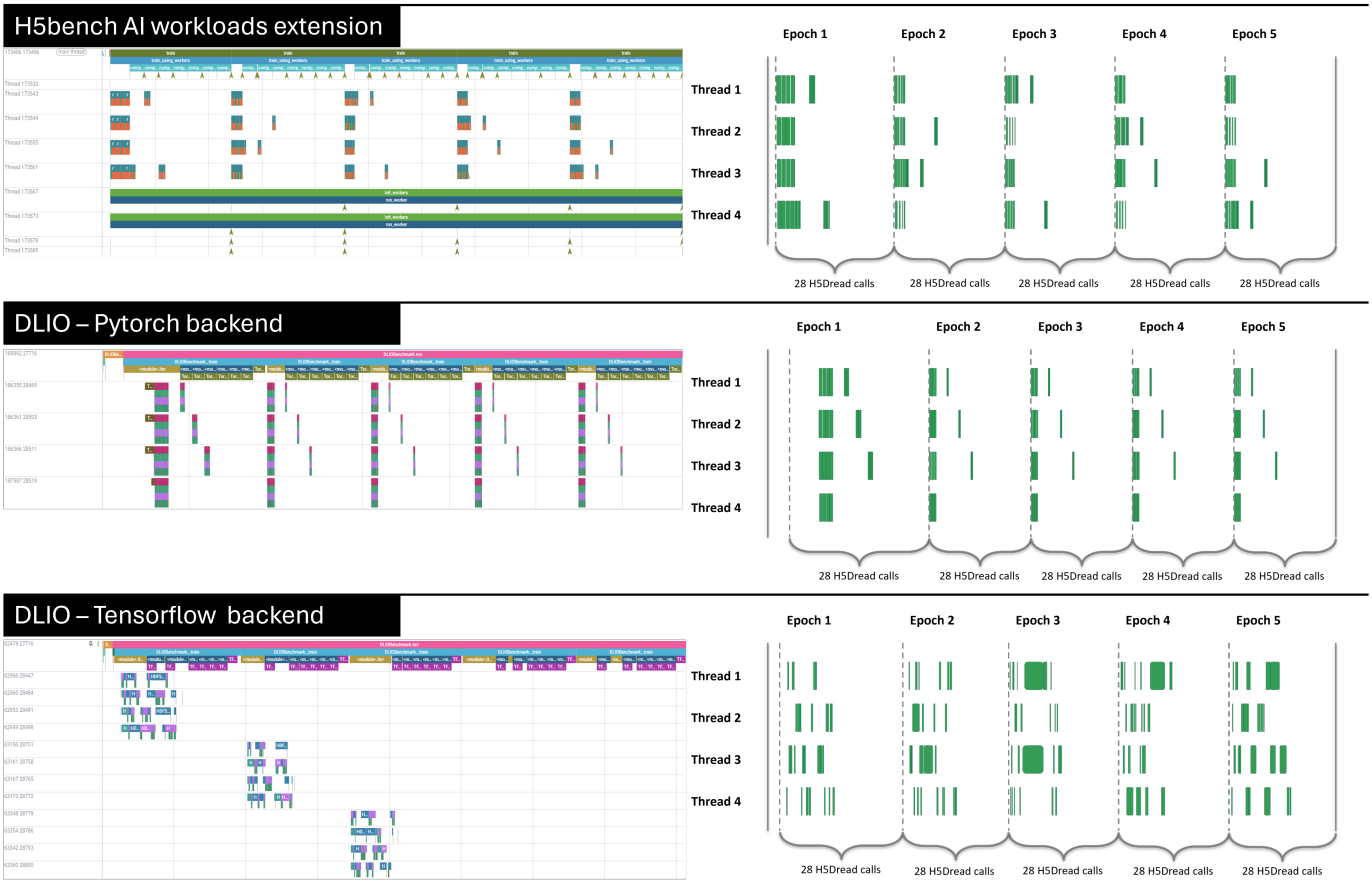

Fig. 2: Trace comparison between DLIO with Pytorch and Tensorflow library, and h5bench extension with the same UNET3D configurations. The left-hand side is DFTracer's trace visualization, and the right-hand side is the extracted `H5Dread` calls from the trace on the left-hand side. The Tensorflow figure is cropped shorter due to the large number of threads spawned based on the number of epochs.

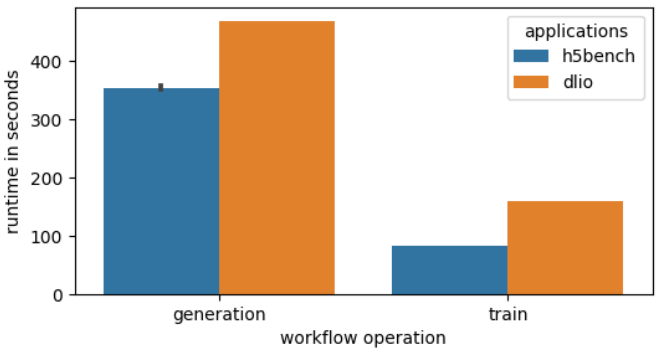

Fig. 3: Runtime comparison between DLIO with Pytorch backend and the h5bench extension for UNET3D in different workflow operations.

An in-depth analysis using DFTracer on DLIO with both Pytorch and Tensorflow backends, as well as our h5bench extension, is presented in Figure 2. This figure provides an overview of how each approach distributes I/O operations across threads. On the left-hand side, we show the traces produced by DFTracer when opened using the Perfetto UI visualization tool[3]. On the right-hand side, we extract and highlight only the `H5Dread` calls from the DFTracer trace shown on the left. Our analysis confirms that each epoch of the UNET3D configuration consistently has 28 `H5Dread` calls.

Our pattern analysis reveals that one factor contributing to the longer runtime is the initialization phase of the PyTorch and TensorFlow libraries. Additionally, we observe that the data loader strategy of each library plays a crucial role in reducing the time required for subsequent I/O operations. Furthermore, the data exchange strategy, which can be strongly influenced by the network and hardware setup, can also significantly impact the I/O time. This impact is particularly evident in the TensorFlow backend, as shown in Figure 2, where it produces new threads per epoch. This behavior affects the duration of some `H5Dread` calls, as seen on the right-hand side of the figure.

To further highlight the variations in I/O duration shown in Figure 2, we performed an addtional test, shown in Figure 4, comparing the DLIO PyTorch backend with our h5bench

[3]https://ui.perfetto.dev/

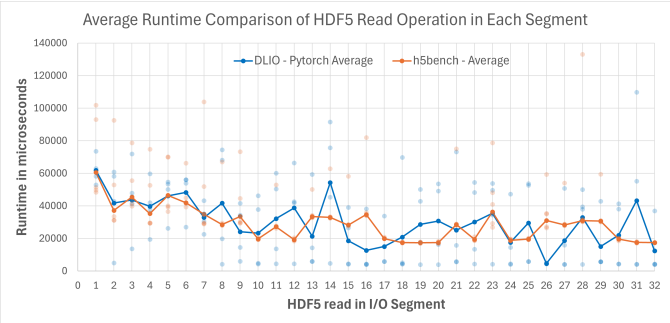

Fig. 4: HDF5 read operation runtime comparison of DLIO and h5bench extension with the same configurations of one process with 8 training files, 4 samples per file, and 4 batch size resulted in 32 read segments in the trace. The experiment is done in the BeeOND filesystem.

extension using identical configurations. In this experiment, calling the HDF5 read function in each segment resulted in varying I/O times while consistently maintaining an overall pattern on average. The primary difference between our extension and the DLIO benchmark lie in the timing of I/O operations, specifically the start and stop times per epoch segment, and the duration of each operation.

B. Scaling Tests with Various Filesystems

We start our test using a single node in BeeOND to gain an initial understanding of workload behavior. For the evaluation, the metrics used are from the original h5bench (raw read rate and observable rate) alongside our cluster's monitoring service (peak fabric bandwidth). The raw read rate is defined as the total amount of read data divided by the time in seconds performing `H5Dread`. The observable rate is calculated as the total read size divided by the difference between the observed time (the total time to complete the benchmark run) and the total compute time of the benchmark.

The read rate in Figure 5 and subsequent figures is based on the values reported by h5bench. This figure shows that the read rate per rank declines as the number of processes increases. This behavior is expected, as the node becomes fully utilized, and the peak network (Fabric) bandwidth is reached with 64 processes. Additionally, the overall read rate reaches its highest point with the maximum number of processes in our test.

When testing the Lustre filesystem at a larger scale (see Figure 6), we observe that the workload scales as the number of processes increases. Notably, there is a jump in the read rate per rank between 96 and 192 processes, which can be attributed to node and network effects.

Finally, comparing the two file systems side by side in Figure 7 reveals that the BeeOND file system in our cluster achieves a higher read rate performance than our Lustre file system. Our h5bench extension runs indicate the need for further optimization of the Lustre file system, particularly as it is mainly used for more traditional HPC workloads.

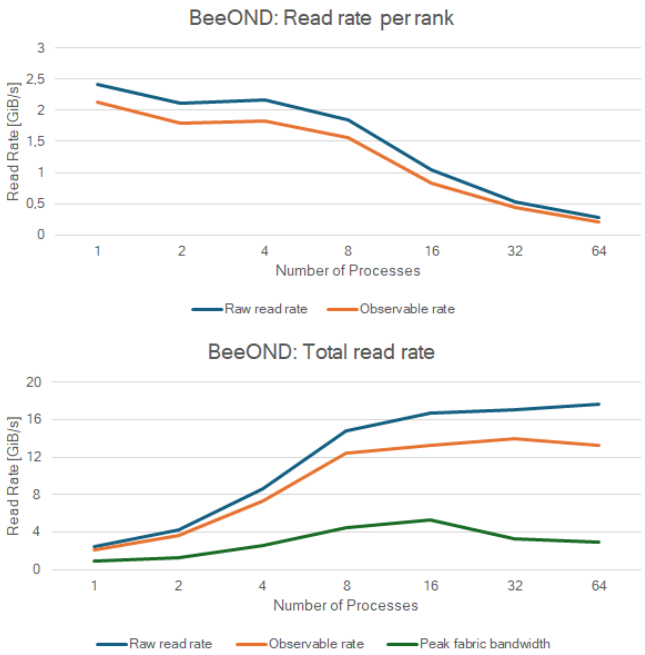

Fig. 5: BeeOND scaling using one node with up to 64 processes.

VIII. Conclusions & Future Work

Our h5bench extension successfully mimics the HDF5 function call patterns of AI workloads, validated through a log VFD comparison of the DLIO runs. In Section VII-A, we analyzed the impact of striping the Python library environment, demonstrating that the underlying HDF5 calls remain consistent. This analysis shows the importance of holistic cluster tuning, specifically the need to isolate I/O HDF5 operations from the effects of the machine learning libraries. In future work, we aim to conduct a more comprehensive analysis examining the impact of the overall network and data transfer strategy introduced by the different machine learning libraries. By extending our study and integrating it with our current work, we want to gain deeper insights into the I/O impact of various file formats, such as TFRecord and npz, compared to the traditional HPC file formats and I/O strategies.

The scaling test confirms that our benchmark works as expected. We successfully ensured that the benchmark suite and configurations effectively test both AI and the traditional HPC workloads already supported by h5bench. With the integration of AI workloads in h5bench, users can streamline the HDF5 benchmarking process for their HPC systems. Additionally, during one of our scaling experiments, we identified the need to tune our Lustre file system for AI workloads.

As our work represents the initial step in incorporating AI workloads in the existing benchmark, we currently only cover a portion of metadata performance, as discussed in Section V-B. Future work can extend this coverage by integrating checkpoint write operations in HDF5, which are not yet used in existing AI workloads but may become relevant for future

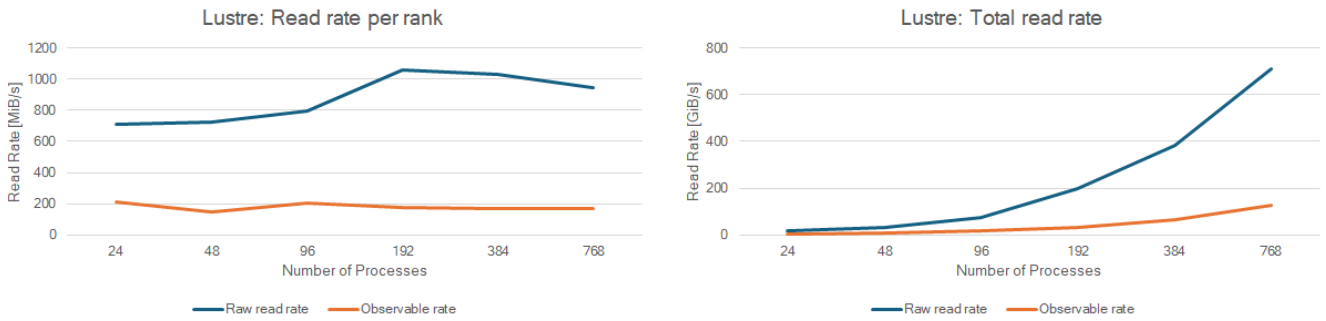

Fig. 6: Workload scaling pattern in Lustre filesystem

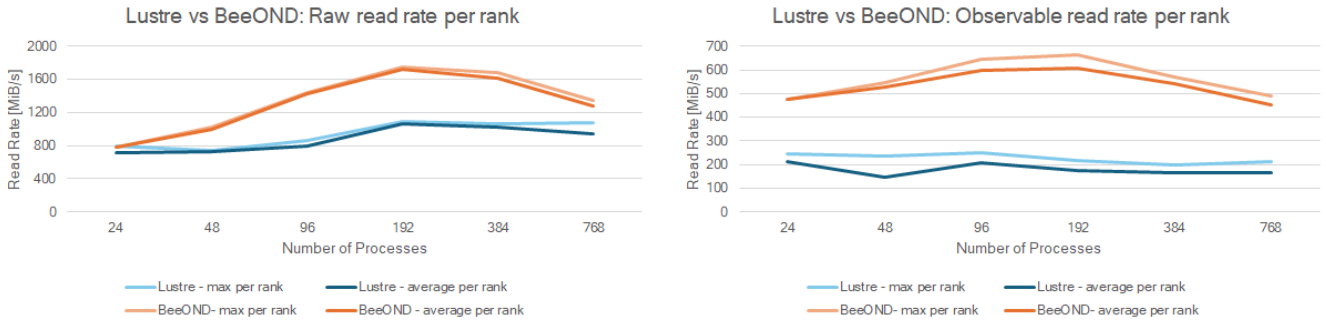

Fig. 7: Comparison of BeeOND and Lustre scaling up to 16 nodes of 24 processes.

AI applications. In addition, features such as asynchronous I/O using the VOL connector and other VOL connector features can enhance the benchmark's utility. Expanding the scope to include workloads from various applications beyond DLIO will provide benchmark users with deeper insights into system performance across a broader range of AI and HPC workloads.

This extension work is currently under review for pull request in: https://github.com/arcturus5340/h5bench

Acknowledgement

Training and evaluation of the machine learning model were performed with computing resources granted by RWTH Aachen University under projects thes1744 and rwth1666. Major parts of this work were produced in the bachelor thesis of Dlyaver Djebarov [38] under the supervision of Radita Liem. We sincerely thank Hariharan Devarajan for his invaluable assistance in running the DLIO and DFTracer and advising us on how to interpret the information, which greatly contributed to the success of this work.

References

[1] Y. Xu, X. Liu, X. Cao, *et al.*, "Artificial intelligence: A powerful paradigm for scientific research," *The Innovation*, vol. 2, no. 4, p. 100 179, 2021, ISSN: 2666-6758. DOI: https://doi.org/10.1016/j.xinn.2021.100179.

[2] F. Su, C. Liu, and H.-G. Stratigopoulos, "Testability and dependability of AI hardware: Survey, trends, challenges, and perspectives," *IEEE Design & Test*, vol. 40, no. 2, pp. 8–58, 2023.

[3] B. Dally, "Hardware for deep learning," in *2023 IEEE Hot Chips 35 Symposium (HCS)*, IEEE Computer Society, 2023, pp. 1–58.

[4] J. L. Bez, S. Byna, and S. Ibrahim, "I/O Access Patterns in HPC Applications: A 360-Degree Survey," *ACM Comput. Surv.*, vol. 56, no. 2, Sep. 2023, ISSN: 0360-0300. DOI: 10.1145/3611007.

[5] S. Neuwirth and A. K. Paul, "Parallel I/O Evaluation Techniques and Emerging HPC Workloads: A Perspective," in *2021 IEEE International Conference on Cluster Computing (CLUSTER)*, 2021, pp. 671–679. DOI: 10.1109/Cluster48925.2021.00100.

[6] H. Devarajan, H. Zheng, A. Kougkas, X.-H. Sun, and V. Vishwanath, "DLIO: A Data-Centric Benchmark for Scientific Deep Learning Applications," no. 81–91, 2021.

[7] O. Balmau, "Characterizing I/O in Machine Learning with MLPerf Storage," vol. 51, no. 3, 2022.

[8] N. Nahar, S. Zhou, G. Lewis, and C. Kästner, "Collaboration challenges in building ML-enabled systems: communication, documentation, engineering, and process," in *Proceedings of the 44th International Conference on Software Engineering*, ser. ICSE '22, Pittsburgh, Pennsylvania: Association for Computing Machinery, 2022, pp. 413–425, ISBN: 9781450392211. DOI: 10.1145/3510003.3510209.

[9] J. L. Bez, H. Tang, S. Breitenfeld, *et al.*, "h5bench: A unified benchmark suite for evaluating HDF5 I/O performance on pre-exascale platforms," *Concurrency and Computation. Practice and Experience*, vol. 36, no. 16, Apr. 2024, ISSN: 1532-0626. DOI: 10.1002/cpe.8046. [Online]. Available: https://www.osti.gov/biblio/2340936.

[10] The HDF Group, *Hierarchical Data Format, version 5*. [Online]. Available: https://github.com/HDFGroup/hdf5.

[11] S. Byna, M. S. Breitenfeld, B. Dong, *et al.*, "ExaHDF5: Delivering Efficient Parallel I/O on Exascale Computing Systems," *Journal of Computer Science and Technology*, vol. 35, no. 1, Jan. 2020, ISSN: 1000-9000. DOI: 10.1007/s11390-020-9822-9. [Online]. Available: https://www.osti.gov/biblio/1582374.

[12] *Introduction - ior 4.1.0+dev documentation*, https://ior.readthedocs.io/en/latest/index.html, [Accessed 08-08-2024].

[13] W. Loewe, R. Hedges, T. McLarty, and C. Morrone, "LLNL's Parallel I/O Testing Tools and Techniques for ASC Parallel File Systems," Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), Tech. Rep., 2004.

[14] J. Kunkel, J. Bent, J. Lofstead, and G. S. Markomanolis, "Establishing the io-500 benchmark," *White Paper*, 2016.

[15] M. Miller, "Multi-Purpose, Application-Centric, Scalable I/O Proxy Application," Lawrence Livermore National Laboratory (LLNL), Livermore, CA (United States), Tech. Rep., 2015.

[16] S. Byna, M. Howison, and A. Sim, "Parallel i/o kernel (piok) suite," *Accessed: Mar*, vol. 16, p. 2020, 2015.

[17] R. W. Houim and E. S. Oran, "A technique for computing dense granular compressible flows with shock waves," *arXiv preprint arXiv:1312.1290*, 2013.

[18] P. Colella, D. Graves, J. Johnson, *et al.*, "Chombo software package for amr applications design document," *Lawrence Berkeley National Laboratory, Berkeley, CA*, 2012.

[19] W. Zhang, A. Almgren, V. Beckner, *et al.*, "AMReX: a framework for block-structured adaptive mesh refinement," *The Journal of Open Source Software*, vol. 4, no. 37, p. 1370, 2019.

[20] V. J. Reddi, C. Cheng, D. Kanter, *et al.*, "MLPerf Inference Benchmark," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020, pp. 446–459. DOI: 10.1109/ISCA45697.2020.00045.

[21] *MLCommons*, https://mlcommons.org/, [Accessed 09-08-2024].

[22] J. Thiyagalingam, M. Shankar, G. Fox, and T. Hey, "Scientific machine learning benchmarks," *Nature Reviews Physics*, vol. 4, no. 6, pp. 413–420, 2022.

[23] T. Ben-Nun, M. Besta, S. Huber, A. N. Ziogas, D. Peter, and T. Hoefler, "A modular benchmarking infrastructure for high-performance and reproducible deep learning," in *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, IEEE, 2019, pp. 66–77.

[24] S. James, Z. Ma, D. R. Arrojo, and A. J. Davison, "Rlbench: The robot learning benchmark & learning environment," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3019–3026, 2020.

[25] C. Coleman, D. Narayanan, D. Kang, *et al.*, "Dawnbench: An end-to-end deep learning benchmark and competition," *Training*, vol. 100, no. 101, p. 102, 2017.

[26] W. Gao, F. Tang, L. Wang, *et al.*, "AIBench: An industry standard internet service AI benchmark suite," *arXiv preprint arXiv:1908.08998*, 2019.

[27] S. W. D. Chien, S. Markidis, C. P. Sishtla, *et al.*, "Characterizing Deep-Learning I/O Workloads in TensorFlow," in *2018 IEEE/ACM 3rd International Workshop on Parallel Data Storage & Data Intensive Scalable Computing Systems (PDSW-DISCS)*, 2018, pp. 54–63. DOI: 10.1109/PDSW-DISCS.2018.00011.

[28] F. Chowdhury, Y. Zhu, T. Heer, *et al.*, "I/O Characterization and Performance Evaluation of BeeGFS for Deep Learning," in *Proceedings of the 48th International Conference on Parallel Processing*, ser. ICPP '19, Kyoto, Japan: Association for Computing Machinery, 2019, ISBN: 9781450362955. DOI: 10.1145/3337821.3337902.

[29] N. Ihde, P. Marten, A. Eleliemy, *et al.*, ""a survey of big data, high performance computing, and machine learning benchmarks"," in *Performance Evaluation and Benchmarking*, R. Nambiar and M. Poess, Eds., Cham: Springer International Publishing, 2022, pp. 98–118, ISBN: 978-3-030-94437-7.

[30] N. Lewis, J. L. Bez, and S. Byna, "I/O in Machine Learning Applications on HPC Systems: A 360-degree Survey," *ACM Comput. Surv.*, Mar. 2025, Just Accepted, ISSN: 0360-0300. DOI: 10.1145/3722215.

[31] Martín Abadi, Ashish Agarwal, Paul Barham, *et al.*, *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*, Software available from tensorflow.org, 2015. [Online]. Available: https://www.tensorflow.org/.

[32] A. Paszke, S. Gross, S. Chintala, *et al.*, "Automatic differentiation in pytorch," 2017.

[33] TensorFlow, *Tfrecord and tf.train.example*, https://www.tensorflow.org/tutorials/load_data/tfrecord, [Accessed 08-08-2024].

[34] NumPy, *Numpy.lib.format*, https://numpy.org/devdocs/reference/generated/numpy.lib.format.html, [Accessed 08-08-2024].

[35] H. Devarajan, L. Pottier, K. Velusamy, *et al.*, "DFTracer: An Analysis-Friendly Data Flow Tracer for AI-Driven Workflows," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, ser. SC '24, Atlanta, GA, USA: IEEE Press, 2024, ISBN: 9798350352917. DOI: 10.1109/SC41406.2024.00023.

[36] Top500, *CLAIX-2023 (CPU segment) - NEC HPC1808Rk-2 DLC, Xeon Platinum 8468 48C 2.1GHz, Infiniband NDR — TOP500 — top500.org*, https://www.top500.org/system/180286/, [Accessed 08-08-2024].

[37] Ö. Çiçek, A. Abdulkadir, S. S. Lienkamp, T. Brox, and O. Ronneberger, ""3d u-net: Learning dense volumetric segmentation from sparse annotation"," in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2016*, S. Ourselin, L. Joskowicz, M. R. Sabuncu, G. Unal, and W. Wells, Eds., Cham: Springer International Publishing, 2016, pp. 424–432, ISBN: 978-3-319-46723-8.

[38] D. Djebarov, "Extending h5bench with I/O access patterns in common AI applications," Veröffentlicht auf dem Publikationsserver der RWTH Aachen University; Bachelorarbeit, RWTH Aachen University, 2024, Bachelorarbeit, RWTH Aachen University, Aachen, 2024, 1 Online–Ressource: Illustrationen. DOI: 10.18154/RWTH-2024-10149.