IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 9 December 2022, accepted 30 December 2022, date of publication 2 January 2023, date of current version 6 January 2023.

Digital Object Identifier 10.1109/ACCESS.2022.3233787

RESEARCH ARTICLE

IOSPReD: I/O Specialized Packaging of Reduced Datasets and Data-Intensive Applications for Efficient Reproducibility

CHAITRA NIDDODI[1], ASHISH GEHANI[2], TANU MALIK[3], SIBIN MOHAN[4], AND MICHAEL LEE RILEE[5], (Member, IEEE)

[1]Department of Computer Science, University of Illinois at Urbana–Champaign, Champaign, IL 61801, USA
[2]SRI International, Menlo Park, CA 94025, USA
[3]School of Computing, DePaul University, Chicago, IL 60614, USA
[4]Department of Computer Science, The George Washington University, Washington, DC 20052, USA
[5]Rilee Systems Technologies LLC, Herndon, VA 20171, USA

Corresponding author: Chaitra Niddodi (chaitra@illinois.edu)

This work was supported in part by the National Science Foundation (NSF) under Grant ACI-1440800 and in part by the Office of Naval Research (ONR) under Contract N68335-17-C-0558.

ABSTRACT The data generated by large scale scientific systems such as NASA's Earth Observing System Data and Information System is expected to increase substantially. Consequently, applications processing these huge volumes of data suffer from lack of storage space at the execution site. This poses a critical challenge while sharing data and reproducing application executions w.r.t. specific user inputs in data-intensive applications. To address this issue, we propose `IOSPReD` (**I/O S**pecialized **P**ackaging of **Re**duced **D**atasets), a data-based debloating framework, designed to automatically track and package only necessary chunks of data (along with the application) in a container. `IOSPReD` uses the specific inputs provided by the user to identify the necessary data chunks. To do so, the high level user inputs are mapped down to low level data file offsets. We evaluate `IOSPReD` on different realistic NASA datasets to assess (i) the amount of data reduction, (ii) the reproducibility of results across multiple application executions and also (iii) the impact on performance.

INDEX TERMS Data management, data-intensive applications, data-based debloating, I/O specialization, containerization, reproducibility.

I. INTRODUCTION

The enormous increase in the rate of data collected by current scientific instruments generates large volumes of data. Per Earth Science Data System (ESDS) approximation, the rate of data input into the Earth Observing System Data and Information System (EOSDIS) is expected to grow annually by massive proportions. Figure 1 shows that this rate of growth will consume exceedingly large volumes of space in the EOSDIS archive by 2025 [5]. The projected increase in the amount of data and the corresponding volume required to store such huge data poses a herculean challenge for applications processing these datasets. This is an example of a system where enormous amounts of data are generated, accessed and processed by scientific applications. Considering the cloud environment that is evolving to accommodate such extensive demand for data assimilation and storage, technologies with scope for data reduction are the need of the hour.

Multi-dimensional scientific data such as meteorological, geophysical data that contain a number of parameters like latitude, longitude, temperature, pressure, humidity usually range in the order of gigabytes in size. Efficient reproducibility of applications is increasingly sought for in the domain of geo-sciences to validate experimental results and their inferences [28]. A typical scientific application consists of two phases — an analysis phase which projects future changes in the environment for improving weather or flood forecasting and a simulation phase which simulates a physical phenomenon such as temperature, pressure, humidity in

The associate editor coordinating the review of this manuscript and approving it for publication was Fabrizio Marozzo.

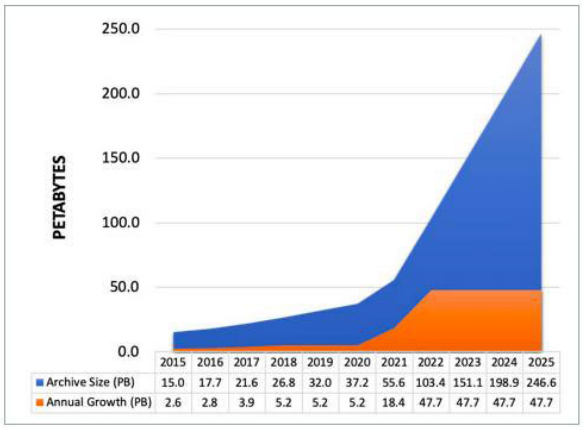


FIGURE 1. Projected Data Increase: Between 2017 and 2025, the volume of data in the EOSDIS archive (blue area) is expected to grow dramatically, accompanied by an order of magnitude increase in the rate of data ingest (orange area). NASA EOSDIS graphic [5].

Earth's land surface using a computational model. The simulation phase usually generates large volumes of data from the computational models, making these scientific applications data-intensive.

A major challenge in reproducing data-intensive scientific applications [36], such as kriging swath data [13], [31], is allocation of available storage space at application execution sites [23]. So, for efficient reproducibility of data-intensive scientific applications and optimal usage of available storage space, data reduction techniques are required.

In order to reproduce application executions w.r.t. specific input scenarios, we do not need the additional irrelevant data that will not be accessed. The question is **can we create a code and data package that has only the necessary data for efficient sharing and reproducibility of data-intensive applications?** To this end, we propose a **data-based debloating** framework, `IOSPReD` (**I/O S**pecialized **P**ackaging of **Re**duced **D**atasets), that *automatically tracks, extracts and stores only the necessary data chunks* along with a data-intensive application. Based on the specific high level inputs provided by the user, `IOSPReD` maps them down to low level data file offsets to identify the necessary data chunks. Then, to access data from the reduced datastore, `IOSPReD` also modifies (*i.e.,* specializes) I/O calls related to data read and write operations. `IOSPReD` finally generates a *complete containerized system that packages the reduced data and specialized library dependencies together with the application* to facilitate efficient sharing and reproducibility of application executions for specific user inputs.

While our framework can be applied to any type of data-intensive application with scope for data reduction, here we focus on scientific applications. Our preliminary work, MiDas [37], used a naive approach for I/O specialization to 'lift' the necessary datachunks into the LLVM bitcode of application libraries. This replaces only the read I/O calls directly with actual datachunks at the callsites while ignoring data redundancy, which could potentially increase the overall data size. `IOSPReD` builds on MiDas and makes the following additional contributions:

1) Identifying and extracting data chunks pertaining to both read and write I/O operations [Section IV-A].
2) Completely eliminating data redundancy during storage by computing file offset intervals so as to extract and store only non-overlapping data chunks [Section IV-B].
3) Support for storing datachunks in two types of datastores — *(i)* LLVM bitcode, *(ii)* a new reduced file [Section IV-C].
4) Support for two different modes of write operation in case of the LLVM bitcode datastore *i.e.,* persistent and non-persistent data writes after each execution of the specialized application [Section IV-E].
5) Support for two modes of specialization *i.e.,* complete specialization of all data files and partial specialization of only certain user-specified data files [Section IV-F].
6) Building a complete containerized system that includes the application along with the specialized library dependencies containing only the necessary data chunks [Section IV-G].

We evaluate our framework on realistic applications across multiple configurations. These applications operate on NASA datasets (in the order of gigabytes) in two different data formats with varying data access patterns to demonstrate the wide range of applicability of `IOSPReD`. Our results show that `IOSPReD` can achieve **upto 97% reduction** on these datasets **while still identifying and including all required datachunks for the provided user inputs to ensure reproducibility of application executions**. Also, our results show that there is no performance degradation while there is a slight speedup of upto 1.004 for debloated (*i.e.,* I/O specialized) applications compared to the original applications.

II. MOTIVATION

Recently, data reduction to efficiently utilize the storage space has gained much focus in the domain of data-intensive computing. However, the currently existing data reduction technique, deduplication, removes only repeating data from a dataset. Our proposed framework, `IOSPReD`, is distinct from deduplication in that the scope of data reduction is beyond eliminating only repeating data. `IOSPReD` extracts and stores a single copy of only necessary data chunks based on given user inputs. These are data chunks accessed by the application while executing these specified user inputs. Rest of the data chunks, duplicate and otherwise, are classified as unnecessary and are removed. Also, the reduced datastore, created as part of `IOSPReD`, does not suffer from the issue of fragmentation, which is a major limitation of deduplication algorithms [32], [43], [45].

Programmers often prefer tools and techniques that alleviate the complexity of programming and help them complete

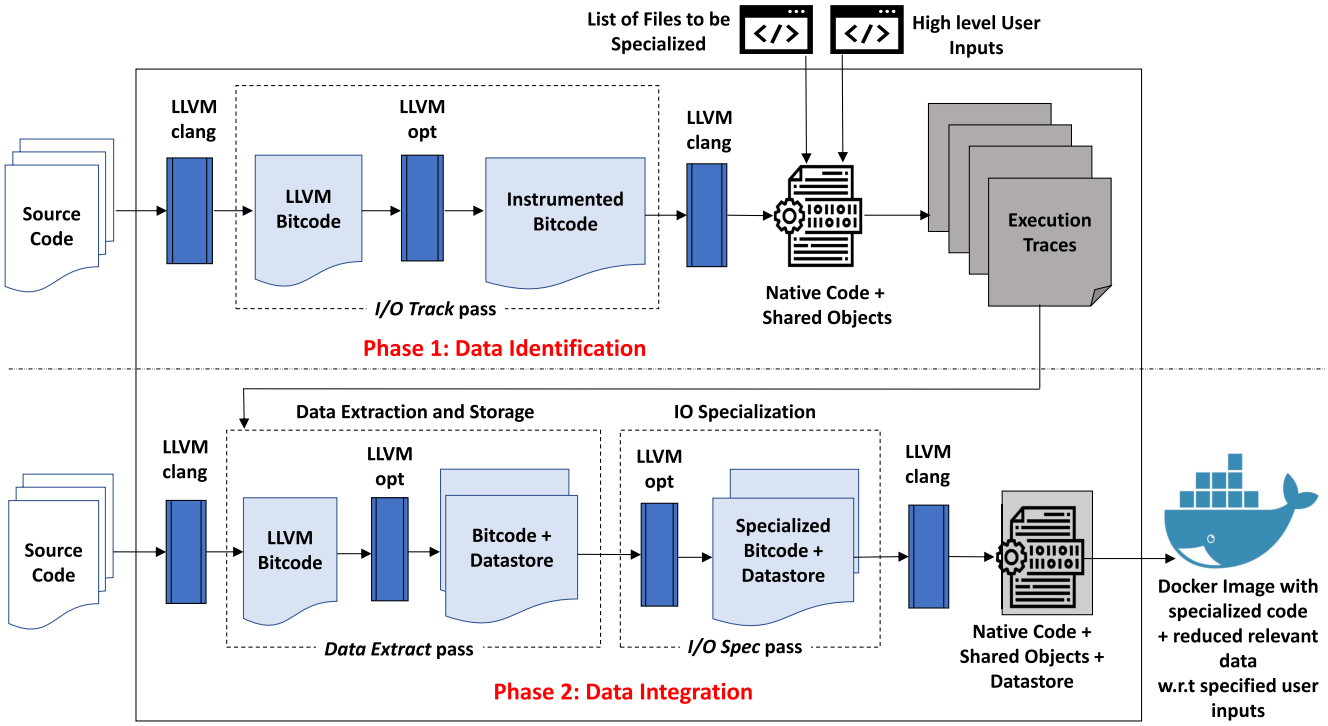


FIGURE 2. IOSPReD: Automated identification and inclusion of relevant datachunks to generate a containerized package with specialized code and necessary data w.r.t specified user inputs.

coding quickly [20], [25]. Standard software libraries designed to manipulate scientific data, *e.g.,* NetCDF4 and HDF5 libraries [9], [16] can be used to generate reduced datasets. However, the challenge lies in reusing the given application with these reduced datasets — users are currently required to manually modify the program to access the reduced datasets correctly. To eliminate the need for manual modification, our tool, `IOSPReD`, automatically rewrites the underlying I/O calls (via I/O specialization) in the program to access the reduced data accurately.

III. BACKGROUND

A. LLVM

We use the LLVM [11] toolchain for compilation and static analysis of code. Its C/C++/Objective-C source frontend, `clang`, is used to produce *bitcode*. The modular LLVM optimizer and analyzer, `opt` is then used to analyze and transform the bitcode. Finally, the transformed bitcode is compiled into native code using `clang`.

B. HDF5 AND NetCDF4 DATA

As mentioned previously, `IOSPReD` is a generalized framework applicable to any data-intensive application with scope for data reduction. In this paper, we look at scientific applications that operate on data in the formats used by NASA Earth Observing System (EOS) — HDF5 [8] and NetCDF4 [15]. HDF5 format uses a 'file directory' like structure to organize data [8]. NetCDF4 stores data in an array-based format [15]. Both these are hierarchical data formats with 'self-describing' data structure. This means that metadata, *i.e.,* data descriptions, are also included along with the data in the file. These formats are used to store various types of data such as climate data, terrain data and geospatial imagery.

IV. IOSPReD

`IOSPReD` automatically tracks, extracts and packages only the necessary data chunks for efficiently reproducing application executions with respect to specific user inputs along with the data-intensive application. This is accomplished in two important phases — **Data Identification** and **Data Integration** as shown in Figure 2. As part of the initial phase, the relevant datachunks are identified by mapping high level user inputs to low level data file offsets. Next, to integrate the data with the application in the second phase, first the datachunks are extracted and stored in an appropriate datastore. Then the application and related libraries are **specialized**, *i.e.,* I/O calls are modified, to access data from the new reduced datastore. Finally `IOSPReD` creates a contanerized package of the reduced datastore and specialized code. The design of `IOSPReD` is *fail-fast*, *i.e.,* when the requested datachunk is not available, the application throws a run time error and aborts execution.

Assumptions. We make the following assumptions: *(i)* The program accesses data in a deterministic manner, *i.e.,* for the same high-level user inputs, the program accesses

data at the same locations across multiple executions. *(ii)* There is only sequential access to data and no parallel access. In particular, when data is shared among multiple processes, we assume that data is manipulated by one process at a time.

Consider a small toy example shown in Figure 3a. This code reads multiple data chunks from *file1.txt* and writes them to *file2.txt*. The file offsets and the corresponding data chunks accessed vary based on the high-level user inputs, *startoff* and *bytes*. We will use this example throughout this section to demonstrate the various steps used as well as the features supported by `IOSPReD`.

A. DATA IDENTIFICATION

The source code containing I/O calls is compiled into LLVM bitcode. We design an LLVM transformation pass, *I/O Track*, to instrument this bitcode as seen in Figure 2. In particular, we add custom functions for relevant I/O calls — `read`, `write`, `open`, `close`, `lseek`, `stat` and their variants in order to **track the file offsets accessed in the data files**. These custom functions take arguments, return values and caller function names corresponding to the I/O calls to log the following attributes: *(i)* file path, *(ii)* start file offset, *(iii)* number of bytes and *(iv)* callsite ID. Callsite ID is a combination of the called function name, the calling function name and a unique integer. This is used to identify the I/O call location in the bitcode.

When I/O calls are directly present in the application, the instrumented bitcode of the application is further compiled into native code. Otherwise, if the I/O calls are part of a library used by the application, then the instrumented bitcode of this library is compiled into a shared object (.so) and integrated with the application. On executing the instrumented application along with inputs specified by the user, execution traces containing the above information (*i.e.,* file names, file offsets *etc.*) are generated. These traces thus identify the necessary datachunks to execute the application for the set of inputs provided by the user. Let's consider the instrumented toy example being executed with four different user inputs – *(i)* 12 and 2, *(ii)* 5 and 3, *(iii)* 10 and 6, *(iv)* 3 and 4. The execution traces generated are shown in Figure 3c and the callsite IDs for the I/O calls to be specialized are shown in Figure 3d.

B. DATA EXTRACTION

We design an LLVM transformation pass, *Data Extract*, to extract and store data chunks in a datastore as shown in Figure 2.

Generating Non-Overlapping File Offsets. The extracted data is based on the offsets generated from the execution traces in the previous step. From these offsets, we compute and store only the overlapping unique data chunks to eliminate data redundancy. This is done using the *sort-merge algorithm*. First the file offset ranges are sorted in ascending order of the start offset. Then, these offset intervals are merged to produce unique offset intervals. Referring back to the toy example, the file offset ranges from the execution traces are 12-14, 5-8, 10-16 and 3-7. After applying merge-sort on them, we get two ranges namely 3-8 and 10-16.

```c
int main(int argc, char **argv) {
 int startoff = atoi(argv[1]);
 int numbytes = atoi(argv[2]);
 int fd, sz;
 char* buf;
 buf = (char*)malloc(nbytes,sizeof(char));
 //file to be specialized
 fd = open("file1.txt", O_RDONLY);
 lseek(fd,startoff,SEEK_SET);
 sz = read(fd, buf, nbytes);
 close(fd);
 //file to be skipped
 fd = open("file2.txt", O_WRONLY);
 sz = write(fd, buf, nbytes);
 close(fd);
}
```

(a) Toy example source code: I/O calls highlighted in red are the ones to be specialized and replaced.

```
This is a sample input text file containing
random text
```

(b) Contents of `file1.txt`.

```
file1.txt 12 2
file1.txt 5 3
file1.txt 10 6
file1.txt 3 4
```

(c) Traces from four executions of the instrumented toy example with various user provided high-level inputs.

```
main;open;1
main;lseek;2
main;read;3
main;close;4
main;open;1
main;lseek;2
main;read;3
main;close;4
main;open;1
main;lseek;2
main;read;3
main;close;4
main;open;1
main;lseek;2
main;read;3
main;close;4
```

(d) Traces from four executions of the instrumented toy example with callsite IDs corresponding to the callsites to be specialized.

```
file1.txt:
[3, 8, "s is a"], [10, 16,"sample "]
```

(e) LLVM Bitcode Datastore: Hashtable with filename as key and list as value containing information related to datachunk — start offset, end offset and actual datachunk in the bitcode datastore.

```
file1.txt:
[3, 8, 0], [10, 16, 6]
```

(f) File Datastore: Hashtable with filename as key and list as value containing information related to datachunk — start offset, end offset in the original file and start offset in the new file datastore.

FIGURE 3. Toy example reading and writing data to files.

C. DATA STORAGE

Next, for efficient storage and retrieval of the extracted datachunks, we use a combination of a hash table with lists. Here the key is the file name and the value is the list containing information related to these data chunks. These are stored as global data structures in the LLVM bitcode.

LLVM Bitcode Datastore. For moderately large datachunks, the LLVM bitcode is chosen as the datastore. In this case, the information in the global data structures include the starting and ending offsets and a pointer to the actual data chunk. For the toy example, the extracted datachunks are stored as shown in Figure 3e.

File Datastore. When the size of datachunks is extremely large (on the order of gigabytes), it is not efficient to store them as part of the LLVM bitcode and load them into RAM. Also, LLVM does not permit extremely large block sizes in bitcode [12]. Therefore, the choice of datastore here is a file. In such a scenario, the information in the global data structures includes the starting and ending offsets with reference to the original data files and the starting offset of the datachunk in the new file. In the toy example, the extracted datachunks are stored as shown in Figure 3f.

In both the above cases, the list is ordered by original file start offset for easy retrieval via linear search.

D. I/O SPECIALIZATION

We design an LLVM transformation pass, *I/O Spec* as shown in Figure 2, to replace I/O calls and their variants to access datachunks stored in the relevant datastore instead of the original data files. It is to be noted that we only replace calls at call sites seen in execution traces identified using unique callsite IDs as described previously. The replaced specialized functions are provided with all arguments of the corresponding original I/O functions. The various I/O calls are replaced as follows:

1) **`open`, `fopen`, `fdopen`, `fileno`, `lseek`, `fseek`, `rewind`, `ftell`, `close`, `fclose` I/O calls and variants.** These are replaced with custom functions that associate the file specific calls with unique integer IDs instead of standard file descriptors or file pointers. These integer IDs are used to access data related information in the global data structures and the corresponding datachunks in the reduced datastore.
2) **`fstat`, `ferror`, `clearerr`, `flock` I/O call and variants.** These are simply replaced with custom functions to return the success value.
3) **`read`, `pread`, `fread`, `write`, `pwrite`, `fwrite` I/O calls and variants.** They are replaced with custom functions designed to access data from the new reduced datastore. To do so, these functions are also provided with a pointer to the global hash table. The filename and file offsets from the original I/O call are matched with the metadata stored as global variables in the bitcode to identify the locations of the datachunks. The corresponding datachunks are then read from or written to the new reduced datastore.

E. MODES OF OPERATION

In reference to the LLVM bitcode datastore, `IOSPReD` supports two modes of operation — *(i)* persistent and *(ii)* non-persistent data writes to the datastore after each execution of the specialized application. User can choose either mode depending on the requirement of the application to be specialized. To ensure the persistent write mode of operation, all new data writes to the data store must be recorded in a log file during execution of the specialized application.

We design an LLVM transformation pass, *Write Persist*, to update datachunks in the bitcode data store using the log file. In particular, this pass deletes all previous global variables related to the datachunks in bitcode and inserts new ones in their place. It creates data structures similar to the *Data Extract* pass in Figure 2, namely a hash table containing file name as key and a list with information pertaining to the data chunks stored.

F. MODES OF SPECIALIZATION

We support two modes of specialization — *(i)* complete specialization of all data files and *(ii)* partial specialization of only certain user-specified data files. The latter is especially useful when an application generates output data files to be used in future. Here, the user would not prefer such data files to be specialized and integrated with the program code. The list of files to be specialized (*i.e.,* non-skip files) is provided by the user to our framework as shown in Figure 2. `IOSPReD` then uses this list to only specialize call sites specific to these non-skip files. Going back to the toy example in Figure 3a, *file1.txt* is an input data file that needs to be specialized (*i.e.,* non-skip file) and *file2.txt* is an output file that must not be specialized, *i.e.,* it must be skipped. Therefore, callsites pertaining to *file1.txt* on lines 8-12 are specialized whereas those on lines 15-17 corresponding to *file2.txt* are not to be specialized (*i.e.,* skip file). This is shown in Figure 3d.

In real world data-intensive applications, more often than not a call site handles all files including skip and non-skip files. For instance, consider the `H5FD_sec2_open` function in HDF5 library. This function opens all HDF5 data files for reading and writing in an application. In such cases, `IOSPReD` first checks whether the file is a skip file or not. If it is not a skip file, then the call is directed to our specialized functions otherwise the call is directed to the original I/O functions.

G. CONTAINERIZATION OF CODE AND DATA

The specialized bitcode of the application is further compiled into native code if I/O calls are directly present in the application. The specialized bitcode of a related library is compiled into a shared object (.so) to be used along with the application in case the I/O calls are part of the library.

After specializing the application and the libraries to store only required data chunks, we finally produce a

Docker container image [2] as the end product (Figure 2) that packages the specialized library dependencies and data together with the application to ensure easier sharing and efficient reproducibility. This Docker container image includes data chunks necessary for application executions for the input configurations provided by the user at the time of specialization to `IOSPReD`. Application executions will succeed as long as high level user inputs map down to access data present in the package's minimized datastore.

Also, note that the containers will work correctly even in case of directory restructuring as filename (and not file path) is the key attribute in identifying datachunks.

V. EVALUATION

Our evaluation intends to address the following research questions:

1) **Research Question 1.** Has `IOSPReD` extracted all the necessary datachunks for the workloads specified by the user?
2) **Research Question 2.** Are the application executions reproducible for the given scenarios?
3) **Research Question 3.** What is the amount of data reduction achieved by `IOSPReD`?
4) **Research Question 4.** Is there a performance degradation?

A. REALISTIC NASA DATASETS

1) EXPERIMENTAL SETUP

We conduct our experiments on a virtual machine with 121GB of RAM, 4-core Intel Xeon(R) CPU E3-1220 v5 at 3.00GHz and running Ubuntu 16.04. We use this system for specialization and creation of a container image with the application, libraries and reduced data. We evaluate `IOSPReD` on the two data formats used by NASA Earth Observing System (EOS) — NetCDF4 and HDF5. The purpose of choosing such datasets is to evaluate if `IOSPReD` can handle varying data access patterns [Appendix VIII-A]. The dataset sizes are chosen so as to demonstrate the purpose of the two datastores supported by `IOSPReD`: LLVM bitcode and reduced file. The HDF5 datasets are moderately large so that the LLVM bitcode can be used as the datastore. On the other hand, the NetCDF4 datasets are too large to be stored as part of the LLVM bitcode and therefore require the use of a file as the datastore.

2) EVALUATION PROGRAMS

Kriging is a generalized technique that can be thought of as an interpolation or prediction scheme for estimating a quantity's spatial distribution using the covariance or variogram of observational data [21]. Remote sensing data is generally irregularly distributed in space and time, leading scientists in general to estimate the value of quantities of interest at points of interest or comparison with theory, simulation, or other observations for integrated analyses. Kriging's ability to adapt to irregular distributions of data, including the calculation of geo-statistical estimates for values in data gaps has led to its use in a wide range of fields, ranging from resource exploration to Earth Science modeling and simulation. The kriging software used in this work was developed for the NASA Open-access Geo-Gridding Infrastructure (NOGGIn) project to provide a way to estimate data values at grid locations from diverse data in a semi-automated fashion [13], [31].

The estimation of a variogram model from observational data is a central and expensive part of the analysis. A variogram model represents how observed properties vary in space (and/or time) as a function of distance from observations. Data is often sampled with enough coverage to capture the important variability in the data, but not so much data as to exceed available computer memory. Finding variogram models that well characterize the variability in observations is generally an iterative process and NOGGIn krige's simple, ad hoc, heuristic variogram fitting is the single most expensive part of the calculation. NOGGIn krige's ad hoc automation is not completely robust, requiring at present some human interaction and reconfiguration and recalculation when variogram fitting fails to converge. Reconfiguring the calculation can involve selecting from a variety of variogram models such as nuggetless gamma-Rayleigh, reassessment of various scale lengths in the data, and the selection of data sampling schemes.

As the kriging depends only on the geometrical overlaps associated with the input data and the output grid, particularly when partitioned for parallel or iterative execution, it should be a good candidate for `IOSPReD`-based optimization. `IOSPReD` speeds up access and loading of just the data being used. As NOGGIn's kriging iterates its variogram fit, `IOSPReD` optimization should be able to reduce unnecessary re-loading of data.

A complicating factor, though, is the set of stochastic sampling methods that are often used, which may impede `IOSPReD`'s cataloging and caching of input data. When such complications are addressed and an observational dataset's variability is well characterized, the variogram model can be stored and reused, ameliorating computational costs, at the expense of the complexity required to re-calculate a new variogram model when the character of the observations change. Once these issues are accommodated, kriging provides a general way to integrate diverse data, *e.g.,* chunks and subsets of data obtained on board spacecraft associated with non-traditional, commercial imagery providers.

Another aspect to be considered in datasets is partitioning or tiling of the data. Each tile is overlaid by a number of data files and different tiles may share (or not) sets of data files. Data is sampled from a larger region than the tile so that the statistics of the variation is more smoothly represented from one tile to the next. Without this extra buffer about the tile the statistics in adjacent tiles would be determined only by data within a tile, leading to a discontinuity in the interpolation at tile boundaries.

3) OMI LEVEL 2 HDF5 DATASET

This is ozone profile (OMO3PR) data from the Ozone Monitoring Instrument (OMI) on the Aura satellite [22]. The datafiles correspond to a swath of observations obtained by the polar orbiting spacecraft on the dayside of the Earth. This data set features vertical profiles of ozone concentration with a spatial resolution of 13 × 48 km (at nadir) and much coarser at the edges (wings) of the swath. EOSDIS datasets are processed at multiple levels ranging from Level 0 to Level 4. Level 0 datasets hold raw data at full instrument resolution. At higher levels, the data is converted into more useful parameters and formats. As a Level 2 dataset, this HDF5 data still retains the spatial layout of the observations made by OMI. That is, the ozone profiles have been registered to geographical coordinates (longitude-latitude), but have *not* been interpolated to a grid, which would make them Level 3 data products. Being able to construct custom Level 3 products involving data fusion or interpolations to interesting, but uncommon grids, as well as checks on existing standard Level 3 products are amongst the capabilities of scientific interest in kriging low-level swath data.

4) VIIRS NetCDF4 DATASET

To explore the impact of `IOSPReD` applied to larger datasets stored in NetCDF4 datafiles, we use kriging on cloud imagery data from the Suomi National Polar-orbiting Partnership (SNPP) Visible Infrared Imaging Radiometer Suite (VIIRS) [14]. This data consists of calibrated and geolocated infrared radiance and reflectance observations with a finest spatial resolution of 375m (at nadir) organized in datafiles corresponding to 6 minutes of operation, resulting in data arrays of 6464 scan lines by 6400 pixels. While it is beyond the scope of this work to go into much detail, VIIRS is a whiskbroom scanner sweeping 32 sensors back and forth across the swath. Therefore, the spatial layout of data arrays is complicated and constructing images requires some processing and motivates our interest in kriging. The VIIRS radiances are considered Level 1 products, in contrast to the Level 2 OMI ozone profiles discussed above, which are derived from other Level 1 OMI data. Thus, the benefits of applying kriging analysis to Level 2 data (like OMI) also applies to Level 1 VIIRS data. While using kriging to fuse data from diverse and irregular spatial distributions is useful, performing statistical covariance or variogram analyses of large numbers of relatively large 40 mega-pixel arrays with irregular spatial distributions can be challenging.

Table 1 summarizes the differences in the datasets being considered. Consequently, the two evaluation programs applying kriging on these datasets vary significantly in terms of handling tiling, 2 dimensional vs. 3 dimensional data, managing memory usage and determining initial parameters for and tuning the variogram fit. There is also greater variability and structure in the VIIRS observation while there is less spatial structure in the OMI data retrieval. The two kriging programs are written in `Python` language. The HDF5 and NetCDF4 I/O libraries are written in `C` language. These underlying libraries are accessed by the kriging programs using `Python` interfaces. This makes automated variogram model fitting harder (*i.e.,* more likely to diverge or poorer fits) for VIIRS. **Here, `IOSPReD`'s data-based debloating approach decreases storage space requirements and aids the iterations required for such analysis.**

B. DATA REDUCTION

1) METRIC

The total size of the required datachunks identified by `IOSPReD` is measured in terms of the unique file offset ranges by computing their sum. The size of `IOSPReD`'s reduced datastore includes the size of all necessary datachunks and the increase in size of the specialized code. We compare this to the total size of the original data files to evaluate the data reduction **[Research Question 3]**.

2) RESULTS

With respect to OMI Level 2 HDF5 Dataset, data is read from 54 HDF5 datafiles of size 578.74 MB. Consider the coordinates lonA=-180, latA=-50, lonB=180, latB=50. We refer to this as configuration **[H]**. The total data accessed by the kriging application is 31.28 MB. There is an overall data reduction of **94.6%** achieved using `IOSPReD`**[Research Question 3.]** as shown in Table 2.

In regard to VIIRS NetCDF4 Dataset, data is read from 480 NetCDF4 datafiles of size 83.34 GB. To show data accesses spanning across the dataset, we use 3 different latitude/longitude configurations at diverse locations on earth, covering varying distances. Consider the coordinates lonA=-165, lonB=-145, latA=10, latB=30 which fall around the region of Hawaii in USA. We refer to this as configuration **[N1]**. The total data accessed by the kriging application in this case is 2.55 GB. There is an overall data reduction of **96.95%** achieved using `IOSPReD`**[Research Question 3.]** as shown in Table 2. For the coordinates lonA=-150, lonB=-110, latA=-60, latB=-20, which we refer to as configuration **[N2]**, the total data accessed by the kriging application is 4.42 GB. There is an overall data reduction of **94.7%** achieved using `IOSPReD`**[Research Question 3.]** as shown in Table 2. With respect to the latitude and longitude values of lonA=20, lonB=107, latA=-10, latB=78, which we refer to as configuration **[N3]**, the total data accessed by the kriging application is 10.7 GB. There is an overall data reduction of **87.16%** achieved using `IOSPReD`**[Research Question 3.]** as shown in Table 2.

C. REPRODUCIBILITY

Note: Successful execution of a specialized application indicates that required datachunks are available. The design is *fail-fast*, *i.e.,* when the requested datachunk is not available,

TABLE 1. Evaluation datasets and configurations.

Dataset	Data size	Number of data files	Number of dimensions	Tile size (lon-lat degrees)	Latitude/Longitude Configurations	IOSPReD Datastore
OMI Level2 HDF5 Dataset	578.74 MB	54	3D (longitude, latitude, altitude)	120x40	H: lonA=-180, latA=-50, lonB=180, latB=50	LLVM bit-code
VIIRS NetCDF4 Dataset	83.34 GB	480	2D (longitude, latitude)	15x15	N1: lonA=-165, lonB=-145, latA=10, latB=30	File
					N2: lonA=-150, lonB=-110, latA=-60, latB=-20	
					N3: lonA=20, lonB=107, latA=-10, latB=78	

TABLE 2. Data reduction in HDF5 and NetCDF4 datasets: The size of IOSPReD's reduced datastore includes the size of all necessary datachunks and the increase in size of the specialized code.

Dataset	Number of Input Data Files	Original Data Size	Dataset Configuration	IOSPReD Reduced Datastore Size	Percentage of Overall Data Reduction
OMI Level2 HDF5 Dataset	54	578.74 MB	H	32.18 MB	94.44%
VIIRS NetCDF4 Dataset	480	83.34 GB	N1	2.55 GB	96.94%
			N2	4.43 GB	94.68%
			N3	10.73 GB	87.13%

the application throws a run time error and aborts execution **[Research Question 1]**.

Metrics. To verify reproducibility of application executions w.r.t specific inputs, we compare the outputs generated in the two cases of application use: *(i)* using original libraries; *(ii)* using the I/O specialized libraries.

It must be noted that `IOSPReD` does not introduce any variation to the outputs generated by the application. Therefore, when the outputs are generated by an application using non-stochastic precise (*i.e.,* deterministic) methods, the outputs in the above two cases exactly match. However, if an application uses stochastic methods (*i.e.,* non-deterministic) then this results in varying outputs being generated across various executions for the same user inputs. In order to establish that `IOSPReD` does not introduce any variation to the outputs generated by an application, we evaluate reproducibility in the following scenarios – when applications generate *(i)* deterministic outputs and *(ii)* non-deterministic outputs.

1) DETERMINISTIC OUTPUTS

We modify the evaluation applications to make them deterministic so as to verify that `IOSPReD` does not introduce any variation to the outputs generated by an application. (Note that these modifications are applied only for this section of the evaluation. For all other evaluation sections before and after this section, we use the default stochastic versions of the applications.) The applications are written in python language. The sources of non-determinism that were modified in the source codes of the applications include: *(i)* APIs in python package `numpy.random` - usage of these APIs were made deterministic by setting `numpy.random.seed(0)`, *(ii)* python API `json.dumps` - the output produced was made deterministic by setting the parameter 'sort_keys=True', *(iii)* accessing dictionaries were made deterministic by iterating over sorted dictionary keys, *(iv)* the variogram estimation model of the Kriging algorithm was made deterministic by setting the parameter 'random_permute=False'.

TABLE 3. Reproducibility of Deterministic Outputs: Output comparison for various dataset configurations: comparing outputs generated by original libraries and I/O specialized libraries.

Dataset Configuration	Number of Output Datafiles	Diff Utility	Difference Reported
H	1	`h5diff`	None
N1	9	`diff`	None
N2	25	`diff`	None
N3	324	`diff`	None

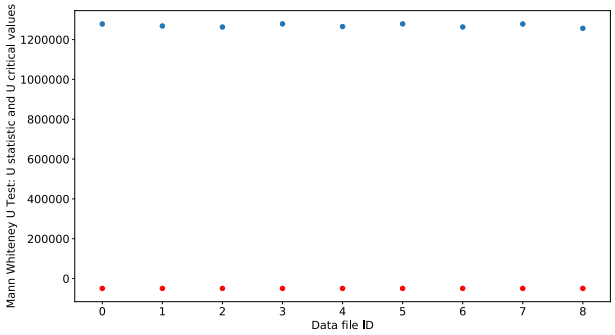
We compare the outputs **[Research Question 2]** as follows:

1) **OMI Level 2 HDF5 Dataset.** We use the standard HDF5 utility, `h5diff` [6], to directly compare the output HDF5 files and report the differences.
2) **VIIRS NetCDF4 Dataset.** We use the standard linux utility, `diff` [1], to directly compare the output NetCDF4 files and check whether or not these exactly match.

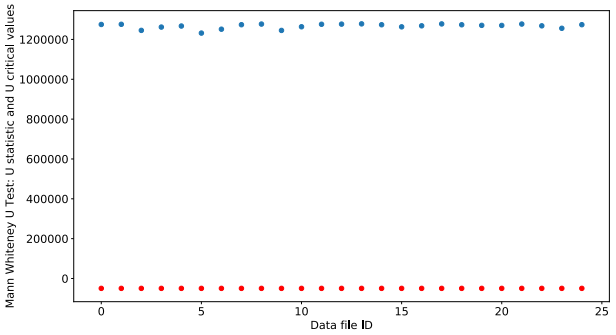
Results. Table 3 shows the results reported by `h5diff` and `diff` respectively. It can be seen that in all cases no difference in reported, *i.e.,* the outputs generated *w.r.t* I/O specialized libraries exactly match the corresponding outputs generated *w.r.t* original libraries. This establishes that `IOSPReD` does not introduce any variation to the outputs generated by an application.

TABLE 4. Reproducibility of Non-Determistic Outputs: Output comparison for various dataset configurations: comparing outputs generated by original libraries and I/O specialized libraries.

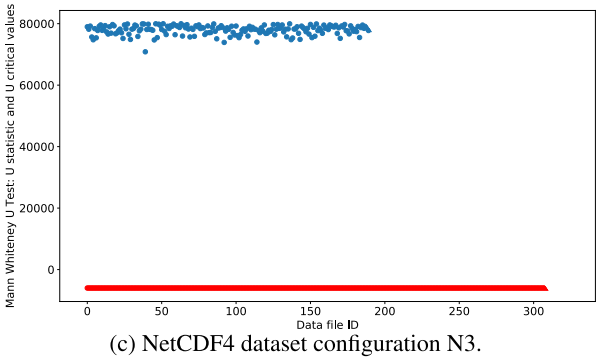
Dataset Configuration	Number of Output Datafiles	U-critical Value	U-statistic Values	Difference Reported by Mann-Whitney U Test	Difference Reported by `h5stat`
H	1	-13244094.88	2102405401.0	None	None
N1	9	-49619.17	in Figure 4a	None	None
N2	25	-49619.17	in Figure 4b	None	None
N3	324	-6005.83	in Figure 4c	None	None



(a) NetCDF4 dataset configuration N1



(b) NetCDF4 dataset configuration N2.



(c) NetCDF4 dataset configuration N3.

FIGURE 4. Reproducibility of Non-Determistic Outputs: Mann-Whitney U Test results for various NetCDF4 dataset configurations: U critical values shown in red and U statistic values shown in blue. In all cases, `U statistic` values are greater than the corresponding `U critical` values [17], [18], [34] which implies that the difference in output files is insignificant.

2) NON-DETERMINISTIC OUTPUTS

The applications we have chosen for evaluation use stochastic sampling methods resulting in varying outputs across various executions for the same user inputs. We compare the outputs **[Research Question 2]** as follows:

1) To compare the high level data statistics such as datatypes, number of values and so on, we use the standard HDF5 utility, `h5stat` [7]. This utility reports statistics of the objects in HDF5 and NetCDF4 datafiles [10]. Then, using the standard linux utility, `diff` [1], a line by line comparison of the outputs generated by `h5stat` is done to check whether these exactly match.
2) To compare the data values we use the *Mann-Whitney U test* [17], [18], [34] to show that the difference between them is statistically insignificant. Mann-Whitney U is a non-parametric test of the null hypothesis which states that for two values X and Y selected randomly from two independent distributions (in our evaluation, the two independent distributions correspond to the two output datafiles being compared), the probabilities of X and Y being greater than each other are equal. We use a two-tailed test with the significance level set to 0.05. Given that the sample size (*i.e.,* number of data values in the output file) is large, standardized value (*i.e.,* z) therefore equals 1.96 [18], [19]. The test statistic 'U' reflects the difference between the two datafiles being compared. We compute `U critical` and `U statistic` values and conclude that the difference is insignificant if `U statistic` is greater than `U critical` [18].

Results. The `U critical` and `U statistic` values for the dataset configurations **H, N1, N2 and N3** are shown in Table 4 and Figure 4. Table 4 also reports the final outcome of Mann Whitney U test and the outcome of `h5stat` comparison. When the number of output files being compared

TABLE 5. Performance comparison: Wall clock run time for a single execution of the application with original libraries vs. that with I/O specialized libraries in seconds and speed up.

Dataset Configuration	Application Execution time with Original libraries	Application Execution time with I/O specialized libraries	Speedup	Size of data accessed
H	160.676 sec	160.414 sec	1.002	0.032 GB of 0.579 GB
N1	6565.564 sec	6543.916 sec	1.003	2.55 GB of 83.34 GB
N2	16049.478 sec	15972.45 sec	1.004	4.43 GB of 83.34 GB
N3	140733.069 sec	140168.57 sec	1.004	10.73 GB of 83.34 GB

is greater than 1, the scatter plots of `U critical` values in red and `U statistic` values in blue are shown in Figures 4a, 4b and 4c. It can be seen that in all cases since the `U statistic` values are greater than the corresponding `U critical` values [17], [18], [34], the difference in the output data values is insignificant. In other words, the outputs generated *w.r.t* the I/O specialized libraries do not differ significantly compared to those *w.r.t* the original libraries.

D. PERFORMANCE

Metric. We use the wall clock execution time in seconds computed for a single execution of the application measured using Python package `time` to evaluate performance **[Research Question 4]**.

Results. Table 5 shows the execution times of the application using the original libraries and that using the I/O specialized libraries for the HDF5 dataset configuration **H** and NetCDF4 dataset configurations **N1, N2, N3**. There is no degradation in performance but rather a slight speedup. With increasing size of data accessed, we see a slight proportionate increase in the speedup of execution times in case of both memory- and filesystem-based specialization datastores, *i.e.,* LLVM bitcode and file datastores respectively.

VI. DISCUSSION

A. USER INPUTS

Our evaluation shows that the reduced datastore produced by `IOSPReD` includes data chunks necessary for application executions w.r.t. specific user inputs provided at the time of specialization. Moreover, it is important to note that application executions will also succeed for other user inputs that map down to access data already present in this minimized datastore. For instance, consider this simple example: a 3D HDF5 data variable stored in an array, 'dv', of size 329 × 30x18. HDF5 data is accessed in chunks. Consider the chunk size is being set to 40 × 30 × 18. If the user input at specialization is dv[10,12,3], the data chunk extracted and packaged is dv[0:39,:,:]. In the reduced container, application executions succeed wrt other inputs such as dv[10,9,3], dv[10,12,15] and dv[39,29,17]. So, the user can tweak high level inputs as long as they fall within the range of the datachunk extracted, *i.e.,* dv[0:39,:,:].

B. PORTABILITY

The end product of `IOSPReD`, *i.e.,* containerized package with specialized code and reduced data, can be easily deployed to reproduce application executions and results. The Docker image is built from a base image corresponding to the operating system of the specialization environment in which the specialized native code/ shared objects are compiled — Ubuntu 16.04 in our case. Therefore, the portability of the Docker container built from such an image is solely affected by the design characteristics of container environments (such as Docker) and not by those of `IOSPReD`. The binaries in the container image will run on a container host if the original and target host kernels share the same *application binary interface (ABI)*. It is not possible to run cross-platform binaries via containers. This also extends to versions of the operating system as well as processor architecture [3], [4].

C. APPLICATION AS TRANSLATOR

We use the application itself as a translator to identify the required data chunks. In particular, given the set of high level user inputs, `IOSPReD` takes a union of all accessed file offset ranges from the various execution traces to compute the necessary data chunks. Our approach of using the application itself as a translator to map from high level user inputs to the underlying file offset ranges is robust enough to handle varying data access patterns with respect to high level user inputs [Appendix VIII-A].

D. DATA FORMAT

`IOSPReD`**maximizes data debloating** by using fine-grained file offset identification to store only the required data chunks in the reduced datastore along with metadata essential for mapping data locations between the original and reduced datastores are present in the reduced datastore. As a result, the reduced datastore holds data in a format different from the original datafile(s) format (HDF5 or NetCDF4).

E. AUTOMATIC MODIFICATION OF PROGRAMS

Programs need not be manually modified by users to access the reduced datastore. `IOSPReD` automatically rewrites the I/O calls (via I/O specialization) in the underlying I/O library programs to access the reduced data accurately.

F. USAGE of HDF5/NetCDF4 DATA ACCESS APIs

Applications can continue to use HDF5/NetCDF4 APIs to access the reduced datastore as if the data was stored in HDF5/NetCDF4 format. This is possible because of I/O specialization — specialized I/O calls in library are

designed to redirect access to the reduced datastore instead of the original HDF5/netcdf datafile. Therefore, application is oblivious to the change of data format in the underlying reduced datastore.

G. SHARING DATA AMONG MULTIPLE APPLICATIONS

Datafiles in HDF5/NetCDF4 formats are usually accessed through applications — *e.g.,* as in the case of kriging applications. Thus, sharing data in these specific formats is not required. Our reduced datastore as part of the specialized library can be shared among multiple applications by including the union of datachunks required by all of these applications in the reduced datastore (assuming that the data in an I/O specialized library is accessed sequentially (not in parallel) by these applications [Assumptions in Section IV]).

H. LIBRARY SEMANTICS

Data extraction in `IOSPReD` is designed to preserve library semantics. Let's say the HDF5 library function `H5FD_sec2_read` expects chunk size of 64 bytes. So, if our specialized I/O functions were to carve out the exact data required and return a chunk size of 59 bytes then the library semantics would be broken. To ensure that this does not happen, `IOSPReD` carves out data chunks in sizes expected by the library.

I. TYPES OF EXTRACTIONS

`IOSPReD` performs two types of extractions — *(i)* extracting only a specific variable in HDF5 and NetCDF4 data files that contain multiple variables such as temperature, pressure and ozone concentration; *(ii)* extracting only specific data chunks within a given variable containing multiple data chunks.

J. COMPARISONS WITH OFF-THE-SHELF UTILITIES

(i) **`strace`.** We compare the LLVM-based instrumentation in `IOSPReD` used for data identification in Phase1 with an off-the-shelf utility, `strace`. `IOSPReD` generates complete traces containing the exact file offsets at which data is accessed along with callsite IDs of the corresponding read/write I/O calls. `strace` generates only partial traces containing the number of bytes read/written (not exact offsets) in read/write I/O calls and also does not trace the callsite IDs of these I/O calls. Table 6 shows that `strace` takes longer to generate just partial execution traces as compared to `IOSPReD` which generates more complete execution traces much faster. This shows that the LLVM-based instrumentation in `IOSPReD` is much more efficient than using `strace`.

(ii) **Sparse Files.** `fallocate` utility in linux can be used to generate sparse files by punching holes at various file offsets. `IOSPReD` achieves higher data reduction compared to generating sparse files. This is because `IOSPReD` performs byte-level reduction whereas there is block-level reduction in sparse files. Moreover, a major advantage of `IOSPReD` is that it allows further specialization of the program due to the file metadata being lifted into the code. Apart from these differences, sparseness of file could be lost if a sparse unaware program tries to write to a sparse file.

TABLE 6. Phase1 Data Identification: Time to generate execution traces - complete traces with `IOSPReD` vs partial traces with `strace`.

Dataset Config.	IOSPReD LLVM-based	`strace`
H	164.731 sec	183.667 sec
N1	6614.680 sec	6835.771 sec
N2	16239.069 sec	16810.631 sec
N3	141112.612 sec	145085.047 sec

VII. RELATED WORK

A. DATA-BASED DEBLOATING

Redundant downloading of data within each layer is a known issue in container-based deployments, resulting in large size images [40]. This has triggered research in the direction of data based debloating in user space. Deduplication is a data reduction technique that focuses only on removing duplicate data. As distinct from this, `IOSPReD` aims to enhance data reduction further by eliminating more than only duplicate data chunks, *i.e.,* `IOSPReD` removes all unnecessary datachunks based on specific user inputs. Fragmentation is a major drawback of deduplication algorithms [32], [43], [45]. In contrast to this, the reduced datastore created by `IOSPReD` does not suffer from fragmentation. A related tool, Slacker [26], uses deduplication of file blocks to create more efficient container systems. However, block-level deduplication techniques do not eliminate data redundancies within structured arrays. Whereas, `IOSPReD` is designed to remove unused data in structured array-based data formats such as in NetCDF4 [15]. Another work, LLIO [39], improves runtime performance of applications through elimination of filesystem accesses. However, it does so by lifting entire files. Such an approach becomes untenable for large data sets. In an attempt to mitigate this limitation, `IOSPReD` demonstrates the possibility of lifting only subsets of data (instead of entire files) and packaging it along with data-intensive applications.

B. CODE-BASED DEBLOATING

Study of code bloat in software with consequent debloating has received a lot of attention in the recent past. Xin et al. [41] have analyzed the tradeoffs between code reduction and function generality in debloated software. Jiang et al. [29], [30] have studied the issue of software bloat in real-world Android applications and have proposed static analysis based techniques to remove dead code. OCCAM [33] and Trimmer [38] have demonstrated that configuration based debloating can be applied to modern applications. Software debloating is currently emerging as a widely adopted technique for security hardening by reducing attack surfaces in code — removing unused pieces of code. μTrimmer [44], designed for MIPS firmware, eliminates unwanted basic blocks in shared libraries. JSLIM [42] is another debloating framework to remove dead code and code containing vulnerabilities in JavaScript applications. Unlike these code-based debloating techniques, `IOSPReD` explores

data-based debloating for efficient storage space utilization and application reproducibility.

C. PROGRAM SPECIALIZATION

A prior work on program specialization by Medicherla et al. [35] has been used to verify whether or not a program 'conforms' to the specified format and to specialize the code to the 'restricted' file format. This is program specialization being used for verification. As opposed to this, program specialization is being used for data reduction in `IOSPReD`. Few other related projects like Decap [27] and C2C [24], in software specialization are mainly focused on reducing code bloat to decrease a software's attack surface. On the contrary, `IOSPReD` specializes code in the context of data debloating.

VIII. CONCLUSION

`IOSPReD` is designed to enable efficient sharing and reproducibility of data-intensive application executions w.r.t specific user inputs. This is done through extraction and packaging of only the relevant data with application code. Our evaluation shows that `IOSPReD` can achieve upto **97% data reduction** in some realistic applications while still being able to ensure **successful reproducibility** of application executions for given input scenarios without causing any **performance degradation**.

APPENDIX

A. VARYING DATA ACCESS PATTERNS

OMI Level 2 HDF5 Datasets. In case of HDF5 datasets, there are varying amounts of data in each value of a dimension due to **data chunking and compression**. The HDF5 library maximizes performance using the mechanisms of specifying how to store data on disk, how to access data, and how to place it in memory. Consider the dataset variable *ozone concentration*. This is 3D data (longitude, latitude, altitude) and thus is a three dimensional array of size 329 × 30x18. The chunk size is 40 × 30x18 and the compression filter being used is *'gzip': 2*. Figure. 5 shows varying amounts of data at different indices of the first dimension of the variable. This is because data is accessed in chunks of size 40 × 30x18. Moreover, the size of data accessed varies across chunks although these chunks sizes are the same, *i.e.,* 40 × 30x18. This is a result of data compression, *'gzip': 2*. The data size is proportional to the number of unique data values within a chunk. In other words, the more the number of unique data values, the larger the size of the data chunk.

VIIRS NetCDF4 Datasets. Consider the dataset variable *I-band 04 earth view radiance*. In contrast to the above 3D HDF5 datasets, this is 2D data (longitude, latitude) and thus is a two dimensional array of size 6464 × 6400. Figure 6 shows varying amounts of data at different indices of the first dimension of this variable. Although the variation in the amount of data across different indices is not as profound as in the previous case of the HDF5 dataset, we do see that not all indices hold the same amount of data.

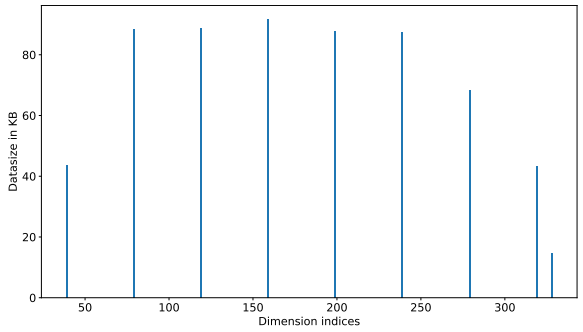


FIGURE 5. Varying amounts of data at each index of the first dimension of variable *ozone concentration* in OMI Level 2 HDF5 Datasets.

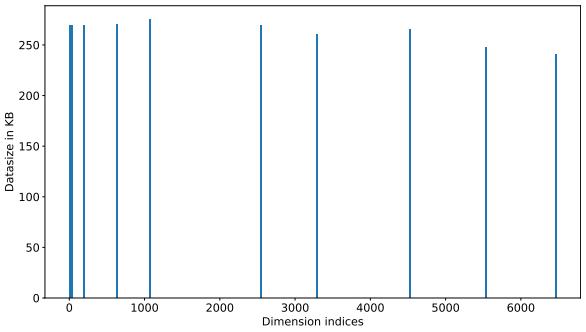


FIGURE 6. Varying amounts of data at each index of the first dimension of variable *I-band 04 earth view radiance* in VIIRS NetCDF4 Datasets.

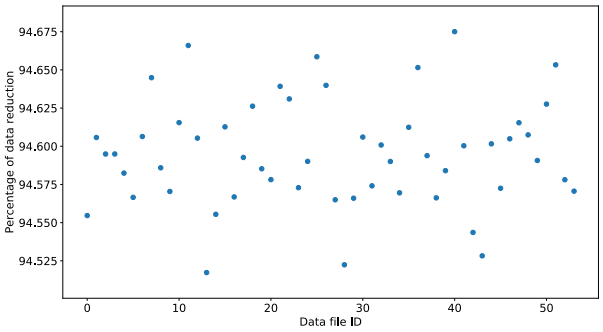


FIGURE 7. The percentage of data reduction in individual input data files: HDF5 dataset configuration H.

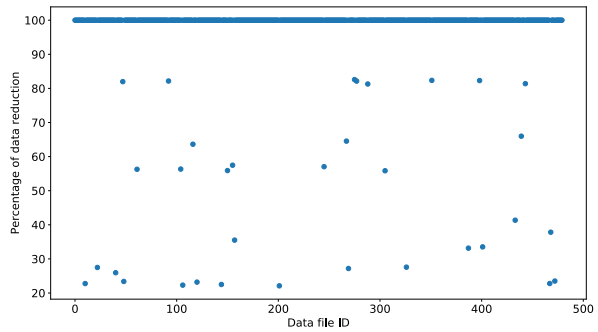


FIGURE 8. The percentage of data reduction in individual input data files: NetCDF4 dataset configuration N1.

Results. Figure 5 and Figure 6 show the variable data access patterns in the HDF5 and NetCDF4 datasets considered in our evaluation. Based on the results in the previous subsections we see that `IOSPReD` is able to accurately identify and extract the required datachunks in these datasets,

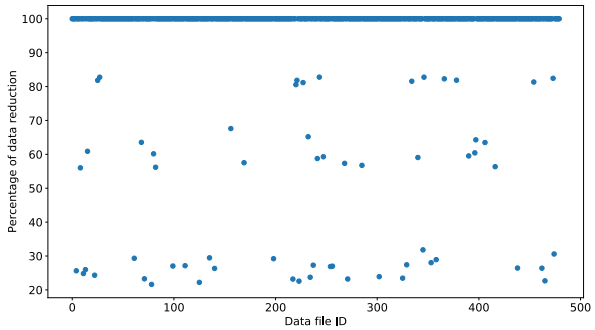


FIGURE 9. The percentage of data reduction in individual input data files: NetCDF4 dataset configuration N2.

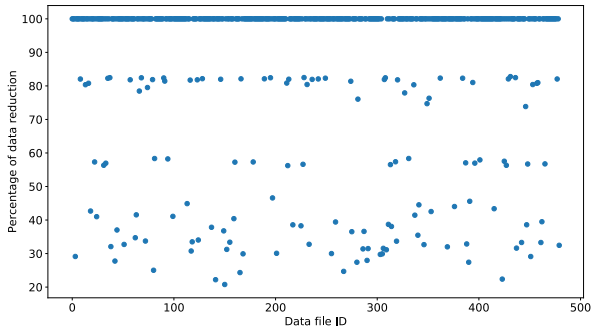


FIGURE 10. The percentage of data reduction in individual input data files: NetCDF4 dataset configuration N3.

thus demonstrating that IOSPReD can handle varying data access patterns.

B. DATA ACCESS IN INDIVIDUAL INPUT DATA FILES

The percentages of data reduction in each of the individual input data files for the various HDF5 and NetCDF4 dataset configurations are shown in Figures 7, 8, 9 and 10.

ACKNOWLEDGMENT

The authors would like to thank Prof. Darko Marinov at University of Illinois at Urbana-Champaign, USA, for providing detailed comments that helped in organizing the article. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF or ONR.

REFERENCES

[1] *Linux Utility Diff*. Accessed: Dec. 2022. [Online]. Available: https://man7.org/linux/man-pages/man1/diff.1.html
[2] *Docker*. Accessed: Dec. 2022. [Online]. Available: https://www.docker.com/
[3] *Docker and OS*. Accessed: Dec. 2022. [Online]. Available: https://sloopstash.com/blog/can-docker-containers-run-on-any-operating-system.html
[4] *Docker Portability*. Accessed: Dec. 2022. [Online]. Available: https://mainframedebate.com/2015/07/16/a-look-at-docker-and-portability/
[5] *Earthdata*. Accessed: Dec. 2022. [Online]. Available: https://www.earthdata.nasa.gov/s3fs-public/imported/EOSDIS_Update_Summer_2017.pdf
[6] *H5STAT*. Accessed: Dec. 2022. [Online]. Available: https://support.hdfgroup.org/HDF5/doc/RM/Tools.html#Tools-Diff
[7] *H5STAT*. Accessed: Dec. 2022. [Online]. Available: https://support.hdfgroup.org/HDF5/doc/RM/Tools.html#Tools-Stat
[8] *HDF5*. Accessed: Dec. 2022. [Online]. Available: https://www.neonscience.org/about-hdf5
[9] *HDF5 APIs*. Accessed: Dec. 2022. [Online]. Available: https://docs.h5py.org/en/stable/quick.html
[10] *HDF5 Tools for NetCDF4 Data*. Accessed: Dec. 2022. [Online]. Available: https://www.unidata.ucar.edu/software/netcdf/workshops/2007/hdf5/ncw07-hdf5.pdf
[11] *LLVM*. Accessed: Dec. 2022. [Online]. Available: https://llvm.org/
[12] *LLVM Malformed Block*. Accessed: Dec. 2022. [Online]. Available: https://lists.llvm.org/pipermail/llvm-commits/Week-of-Mon-20150518/277034.html
[13] *NASA/ACCESS-15 NOGGIN*. Accessed: Dec. 2022. [Online]. Available: https://github.com/michaelleerilee/NOGGIN
[14] *VIIRS*. Accessed: Dec. 2022. [Online]. Available: https://ladsweb.modaps.eosdis.nasa.gov/missions-and-measurements/products/VNP02IMG, doi: 10.5067/VIIRS/VNP02IMG.002.
[15] *NetCDF4*. Accessed: Dec. 2022. [Online]. Available: https://www.earthdatascience.org/courses/use-data-open-source-python/hierarchical-data-formats-hdf/intro-to-climate-data/
[16] *NetCDF4 APIs*. Accessed: Dec. 2022. [Online]. Available: https://unidata.github.io/netcdf4-python/
[17] *Nonparametric Tests*. Accessed: Dec. 2022. [Online]. Available: https://corporatefinanceinstitute.com/resources/knowledge/other/nonparametric-tests/
[18] *Statistical Significance*. Accessed: Dec. 2022. [Online]. Available: http://users.sussex.ac.uk/~grahamh/RM1web/Wilcoxon%20Large%20N%202009.pdf
[19] *Statistical Z Value*. Accessed: Dec. 2022. [Online]. Available: https://en.wikipedia.org/wiki/Standard_score
[20] J. Brandt, P. J. Guo, J. Lewenstein, M. Dontcheva, and S. R. Klemmer, "Opportunistic programming: Writing code to prototype, ideate, and discover," *IEEE Softw.*, vol. 26, no. 5, pp. 18–24, Sep. 2009.
[21] J.-P. Chilés and P. Delfiner, *Geostatistics Modeling Spatial Uncertainty*. Hoboken, NJ, USA: Wiley, 2012.
[22] J. D. Haan and P. Veefkind, "OMI/Aura ozone (O3) profile 1-orbit L2 swath 13 × 48 km V003," Goddard Earth Sci. Data Inf. Services Center (GES DISC), Greenbelt, MD, USA, Tech. Rep., 2009. [Online]. Available: https://disc.gsfc.nasa.gov/datasets/OMO3PR_003/summary, doi: 10.5067/Aura/OMI/DATA2026.
[23] E. Deelman and A. Chervenak, "Data management challenges of data-intensive scientific workflows," in *Proc. 8th IEEE Int. Symp. Cluster Comput. Grid (CCGRID)*, May 2008, pp. 687–692.
[24] S. Ghavamnia, T. Palit, and M. Polychronakis, "C2C: Fine-grained configuration-driven system call filtering," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, New York, NY, USA, 2022, pp. 1243–1257, doi: 10.1145/3548606.3559366.
[25] J. P. Guo and D. Engler, "Towards practical incremental recomputation for scientists: An implementation for the Python language," in *Proc. Workshop Theory Pract. Provenance*, 2010, pp. 1–31.
[26] T. Harter, B. Salmon, R. Liu, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Slacker: Fast distribution with lazy Docker containers," in *Proc. 14th USENIX Conf. File Storage Technol. (FAST)*, 2016, pp. 181–195.
[27] M. M. Hasan, S. Ghavamnia, and M. Polychronakis, "Decap: Deprivileging programs by reducing their capabilities," in *Proc. 25th Int. Symp. Res. Attacks, Intrusions Defenses*, Oct. 2022, pp. 395–408, doi: 10.1145/3545948.3545978.
[28] F. Hoffeins, F. M. Ciorba, and I. Banicescu, "Examining the reproducibility of using dynamic loop scheduling techniques in scientific applications," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops (IPDPSW)*, May 2017, pp. 1579–1587, doi: 10.1109/IPDPSW.2017.147.
[29] Y. Jiang, D. Wu, and P. Liu, "JRed: Program customization and bloatware mitigation based on static analysis," in *Proc. IEEE 40th Annu. Comput. Softw. Appl. Conf. (COMPSAC)*, Jun. 2016, pp. 1–21.
[30] Y. Jiang, Q. Bao, S. Wang, X. Liu, and D. Wu, "RedDroid: Android application redundancy customization based on static analysis," in *Proc. IEEE 29th Int. Symp. Softw. Rel. Eng. (ISSRE)*, Oct. 2018, pp. 189–199.
[31] K.-S. Kuo, R. Wolfe, A. Radov, and T. Clune, "Noggin-NASA open-access geo-gridding infrastructure," in *Proc. AMS 99th Annu. Meeting*, 2019. [Online]. Available: https://www.researchgate.net/publication/341255009_NOGGIN_-_NASA_OPEN-ACCESS_GEO-GRIDDING_INFRASTRUCTURE#fullTextFileContent, doi: 10.13140/RG.2.2.11140.35209.
[32] L. Lin, Y. Deng, Y. Zhou, and Y. Zhu, "Inde: An inline data deduplication approach via adaptive detection of valid container utilization," *ACM Trans. Storage*, Nov. 2022. [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/3568426, doi: 10.1145/3568426.

[33] G. Malecha, A. Gehani, and N. Shankar, "Automated software winnowing," in *Proc. 30th Annu. ACM Symp. Appl. Comput.*, Apr. 2015, pp. 1504–1511.
[34] H. B. Mann and D. R. Whitney, "On a test of whether one of two random variables is stochastically larger than the other," in *Proc. Ann. Math. Statist.*, vol. 18, no. 1, pp. 50–60, 1947.
[35] R. K. Medicherla, R. Komondoor, and S. Narendran, "Program specialization and verification using file format specifications," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol. (ICSME)*, Sep. 2015, pp. 191–200, doi: 10.1109/ICSM.2015.7332465.
[36] H. Meng, R. Kommineni, Q. Pham, R. Gardner, T. Malik, and D. Thain, "An invariant framework for conducting reproducible computational science," *J. Comput. Sci.*, vol. 9, pp. 137–142, Jul. 2015.
[37] C. Niddodi, A. Gehani, T. Malik, J. A. Navas, and S. Mohan, "MiDas: Containerizing data-intensive applications with I/O specialization," in *Proc. 3rd Int. Workshop Practical Reproducible Eval. Comput. Syst.*, Jun. 2020, pp. 21–25.
[38] H. Sharif, M. Abubakar, A. Gehani, and F. Zaffar, "TRIMMER: Application specialization for code debloating," in *Proc. 33rd ACM/IEEE Int. Conf. Automated Softw. Eng.*, Sep. 2018, pp. 329–339.
[39] C. Smowton, "I/O optimisation elimination via partial evaluation," Ph.D. thesis, Comput. Lab., Univ. Cambridge, Cambridge, U.K., 2014. [Online]. Available: https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-865.pdf
[40] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at Google with Borg," in *Proc. 10th Eur. Conf. Comput. Syst.*, Apr. 2015, pp. 1–17.
[41] Q. Xin, Q. Zhang, and A. Orso, "Studying and understanding the tradeoffs between generality and reduction in software debloating," in *Proc. 37th IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Oct. 2022, pp. 1–13.
[42] R. Ye, L. Liu, S. Hu, F. Zhu, J. Yang, and F. Wang, "JSLIM: Reducing the known vulnerabilities of Javascript application by debloating," in *Emerging Information Security and Applications*, W. M. Sokratis and K. Katsikas, Eds. Cham, Switzerland: Springer, 2022, pp. 128–143.
[43] D. Zhang, Y. Deng, Y. Zhou, Y. Zhu, and X. Qin, "Improving the performance of deduplication-based backup systems via container utilization based hot fingerprint entry distilling," *ACM Trans. Storage*, vol. 17, no. 4, pp. 1–23, Oct. 2021, doi: 10.1145/3459626.
[44] H. Zhang, M. Ren, Y. Lei, and J. Ming, "One size does not fit all: Security hardening of MIPS embedded systems via static binary debloating for shared libraries," in *Proc. 27th ACM Int. Conf. Architectural Support Program. Lang. Operating Syst.* New York, NY, USA, Feb. 2022, pp. 255–270, doi: 10.1145/3503222.3507768.
[45] N. Zhao, H. Albahar, S. Abraham, K. Chen, V. Tarasov, D. Skourtis, L. Rupprecht, A. Anwar, and R. A. Butt, "DupHunter: Flexible high-performance deduplication for Docker registries," in *Proc. USENIX Annu. Tech. Conf. (USENIX)*. Berkeley, CA, USA: USENIX Association, Jul. 2020, pp. 769–783, [Online]. Available: https://www.usenix.org/conference/atc20/presentation/zhao

CHAITRA NIDDODI received the B.E. degree in computer science from PES University, India, and the M.S. degree in computer science from the University of Illinois at Urbana–Champaign, where she is currently pursuing the Ph.D. degree with the Department of Computer Science. Her research interests include code analysis and rewriting for data debloating and improved fuzz testing.

ASHISH GEHANI received the B.S. degree in mathematics from the University of Chicago and the Ph.D. degree in computer science from Duke University. He is currently a Senior Principal Computer Scientist with SRI International. His research interests include data provenance and security.

TANU MALIK is currently an Associate Professor with the School of Computing, DePaul University. Her research interests include scientific data management, data provenance management, data virtualization techniques for reproducible science, caching, optimization, and approximations in large data.

SIBIN MOHAN received the bachelor's degree in computer science and engineering from Bangalore University, India, and the M.S. and Ph.D. degrees in computer science from North Carolina State University. He is currently an Associate Professor with the Department of Computer Science, The George Washington University. His research interests include systems, security, networking, and autonomous systems.

MICHAEL LEE RILEE (Member, IEEE) is currently with Rilee Systems Technologies LLC. His research interests include big data, geographic information systems, and geophysics computing.