PARLO: *PA*rallel *R*un-time *L*ayout *O*ptimization for Scientific Data Explorations with Heterogeneous Access Patterns

Zhenhuan Gong[1,2], David A. Boyuka II[1,2], Xiaocheng Zou[1,2], Qing Liu[2], Norbert Podhorszki[2], Scott Klasky[2], Xiaosong Ma[1,2], Nagiza F. Samatova[1,2,*]

[1]North Carolina State University, Raleigh, NC 27695, USA
[2]Oak Ridge National Laboratory, Oak Ridge, TN 37831, USA
[*]Corresponding author: samatova@csc.ncsu.edu

***Abstract*—The size and scope of cutting-edge scientific simulations are growing much faster than the I/O and storage capabilities of their run-time environments. The growing gap is exacerbated by exploratory, data–intensive analytics, such as querying simulation data with multivariate, spatio-temporal constraints, which induces *heterogeneous access patterns* that stress the performance of the underlying storage system. Previous work addresses data layout and indexing techniques to improve query performance for a single access pattern, which is not sufficient for complex analytics jobs. We present PARLO a parallel run-time layout optimization framework, to achieve multi-level data layout optimization for scientific applications at run-time before data is written to storage. The layout schemes optimize for heterogeneous access patterns with user-specified priorities. PARLO is integrated with ADIOS, a high-performance parallel I/O middleware for large-scale HPC applications, to achieve user-transparent, light-weight layout optimization for scientific datasets. It offers simple XML-based configuration for users to achieve flexible layout optimization without the need to modify or recompile application codes. Experiments show that PARLO improves performance by 2 to 26 times for queries with heterogeneous access patterns compared to state-of-the-art scientific database management systems. Compared to traditional post-processing approaches, its underlying run-time layout optimization achieves a 56% savings in processing time and a reduction in storage overhead of up to 50%. PARLO also exhibits a low run-time resource requirement, while also limiting the performance impact on running applications to a reasonable level.**

I. Introduction

Extreme-scale, multivariate, spatio-temporal datasets from high-fidelity scientific simulations present great challenges for knowledge discovery largely due to I/O bottlenecks. Optimizations for storage and I/O systems in HPC have been actively addressed at multiple levels of the software stack — from parallel file systems (PFS) (e.g., GPFS [18], Lustre [6], PVFS [3]) through I/O middleware (e.g., ADIOS [14], HDF5 [7], Parallel netCDF [13]), to data staging architecture (e.g., DataStager [1], PreDatA [26]). However, such optimizations have been primarily driven by the need for fast data offloads from the simulation run-time environment by maximizing *write* throughput.

In contrast, *read performance* for exploratory queries and analytics has not been addressed with the same level of attention. Whereas previously there was no significant issue in this area, the growing size and complexity of scientific datasets are beginning to overwhelm simple approaches, creating an *analysis bottleneck*. The root cause is that many analytics tend to induce highly heterogeneous and hard-to-predict access patterns over the data, whereas coarse-grained data storage method can only efficiently serve one such pattern. Developing new layout optimization (LO) methods for organizing data on disk is crucial to addressing this rising issue.

Previous work has demonstrated a number of layout optimization techniques, but each for only a particular access pattern. For instance, SciDB [2] and work on space-filling curves (SFC) [17] focus on spatial LO. Likewise, FastBit [5], [25] and ISABELA-QA [16] explore value-based LO methods. However, systems optimized for only a single access pattern cannot address the mix of access patterns observed in practice. While simply generating multiple replicas of a dataset, each with a different LO, may seem a straightforward solution, it is infeasible under the storage constraints of current HPC systems, and would also have a substantial, negative impact on effective simulation I/O write throughput, which is still a critical issue. Instead, an efficient, holistic LO scheme should be able to optimize for heterogeneous access patterns while avoiding such replication.

Another limitation of existing LO techniques is the absence of a run-time framework capable of applying LO over datasets at application run time, before they are written to storage. Instead, most previous work considers only post-processing methods, which necessarily entail the overhead of reading, processing, and writing entire datasets. Such an approach is becoming prohibitively expensive in most cases due to the ubiquitous I/O bottleneck. Some previous work does address LO at file system level [20], [23]; however, these methods require application-level knowledge, and only optimize for specific access patterns and file systems.

In response to these limitations of existing systems, we present PARLO, a parallel run-time layout optimization framework. PARLO extends our previous work, MLOC [10], to optimize for heterogeneous access patterns on scientific datasets both through parallel run-time LO and integration with the ADIOS adaptive I/O middleware. Users may configure PARLO to optimize for different access patterns via the standard ADIOS XML configuration file without modifying or recompiling application code. Additionally, we have extended the ADIOS read APIs to offer a rich query interface, offering improved query performance under heterogeneous access patterns by leveraging our LO schemes.

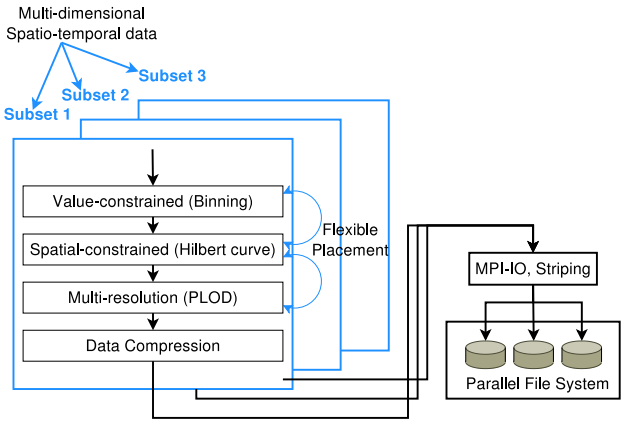


Fig. 1. An overview of MLOC's multi-level architecture. The positions of levels for layout optimization (LO) are flexible.

This paper makes the following contributions:

- We present PARLO, a parallel run-time layout optimization framework for queries on large complex scientific datasets with heterogeneous access patterns.
- PARLO is integrated with ADIOS middleware, and can be used flexibly and transparently by existing user applications. In keeping with ADIOS common practice, PARLO optimization parameters can be tuned via the XML configuration file without recompilation.
- PARLO adds rich query interfaces to the middleware read APIs to fully leverage dataset LO to improve query performance by 2 to 26 times.
- PARLO is designed and implemented as a lightweight plug-in to minimize run-time performance impact on large-scale parallel HPC applications. The additional run-time overhead is between 3% and 30% for both run-time layout optimization and query index building. Compared to traditional post-processing approaches, it saves processing time by 56% and storage overhead by 50%.

The remainder of this paper is organized as follows: Section II introduces important background related to our work. Section III describes the design and implementation details of PARLO. Section IV shows experimental evaluation for PARLO, while Section V gives a broader comparison with existing related work. Finally, Section VI concludes and summarizes our work.

II. Background

A. MLOC: Multi-level Layout Optimization for Compressed Scientific Datasets

In our previous work, we have presented MLOC, a multi-level layout framework for compressed, scientific, multivariate, spatio-temporal datasets. MLOC provides a flexible multi-level architecture in which multiple layout optimizations are composed and ordered according to user-specified priority to optimize for heterogeneous access patterns. MLOC also includes a byte-level method for precision-driven multi-resolution data access, which achieves higher detail preservation than traditional multi-resolution sampling. Data compression techniques can be selectively plugged into MLOC to reduce storage and I/O overhead. Experiments have shown that MLOC achieves higher query performance and lower storage overhead compared to the state-of-the-art techniques [10].

B. I/O Middleware and Run-time Data Transformation

Because handling parallel I/O in an HPC system is a complex task, many modern scientific applications use publically available I/O middleware solutions, such as HDF5 [7], ADIOS [14], and PnetCDF [13], to manage this process. Such libraries provide stable and portable I/O interfaces, allow scientists to leverage existing best practices for performing I/O, and produce data in a self-describing file format.

Although, these systems deliver efficient I/O performance, there is sparing support for run-time data transformation methods, such as is our goal to provide in this paper. HDF5, for instance, has implemented compression filters for run-time data compression. However, this mechanism has two main limitations that prevent us from using it in this work. First, the filters only support compression-like algorithms that reduce data size, and do not offer enough flexibility for developers to achieve more versatile functionality (such as an added index, or the ability to read only a portion of the transformed data to answer a query). Second, the filters do not currently work in parallel write mode, as the compressed buffer sizes are unpredictable, and thus make it difficult for the middleware to calculate file write offsets without expensive global synchronization. This is unacceptable for our application, as we wish to target modern large-scale parallel scientific codes with our run-time layout optimization, so filters are not an option.

Given these limitations, and the fact that other major I/O middleware codes (ADIOS and pnetCDF) do not support a mechanism like filters, we found it necessary to extend an I/O middleware to support our run-time data transformation. In this work, we integrate with the ADIOS I/O library; the specifics are described in Section III-B. Next, we give an overview of ADIOS, and why it is a good fit as a platform with which to integrate our work.

C. ADIOS I/O Middleware and BP File Format

ADIOS is an I/O middleware library that aims to provide an efficient parallel I/O pipeline for large-scale scientific simulations. Key features include highly scalable reading and writing; a componentized I/O stack; an extensible, portable, and metadata-rich binary file format; and delayed consistency writes to minimize global synchronization.

Several of these features of ADIOS have a direct impact on integration with PARLO, and so require further explanation. First, ADIOS's componentization of the HPC I/O stack abstracts and separates the read and write transport methods from the core ADIOS "common layer". This offers a centralized place for us to intercept the user's data to optimize its storage layout. Second, ADIOS uses a configuration XML file, which contains variable names, groupings, datatypes, and dimensions (which can be parameterized at run time). We extend this

configuration as we integrate PARLO by allowing the user to mark individual variables for particular layout optimizations using a simple new parameter (as exemplified in Figure 4).

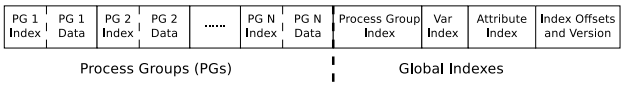


Fig. 2. The structure and ADIOS BP file format.

One more relevant aspect of ADIOS that must be understood is its current native storage layout, the Binary Packed (BP) file format. A high-level overview of the layout is given in Figure 2. The fundamental data storage unit of a BP file is the Process Group (PG), which contains all of the variable data written by a single process during a single write phase, and includes the necessary metadata to describe its contents. When writing is finished, basic metadata for all PGs is then collected and appended to the BP file as a footer index (the Global Index section shown in the figure). This format is important here because PARLO performs both *intra-* and *inter-PG* optimizations. The former independently and locally transforms the data produced by each processor, whereas the latter reorders the PGs themselves in the file to increase spatial locality. These techniques are addressed in detail in Section III-C.

III. Method

The overall goal of PARLO is to optimize the data layout for multi-dimensional, spatio-temporal scientific datasets in order to improve query performance under a range of different access patterns. PARLO targets several classes of query-driven access patterns induced by different kinds of analytics jobs: (1) value-constrained access patterns, generated by value queries. For example, range queries, histogram building, isosurface construction, and so on; (2) spatial/region-constrained access patterns, generated by range/region queries. For example, multi-dimensional array slicing, subvolume/subplane access, etc.; (3) value-and-spatial-constrained access patterns, resulting from a combination of the previous two access patterns; and (4) multi-resolution data access patterns, induced by partial-precision/approximate queries. For example, visualizing data at a lower value precision to reduce I/O time, using reduced-precision data to speed up simple analysis, and so on. More detailed characterizations of these query types can be found in our previous work [9], [10].

Pursuant to this broad goal, and given the limitations of current approaches outlined previously, in PARLO we identify the following specific goals:

- Design and implement a parallel run-time framework to achieve LO at run time instead of post-processing, saving computation, I/O and storage resources.
- Integrate the run-time framework with I/O middleware to achieve user-transparent LO without modification or recompilation of application codes.
- Reduce run-time overhead LO to minimize the performance impact on running applications.
- Provide rich query functions for the middleware read APIs, and optimize the query processing for heterogeneous access patterns.

A. Approach Overview

PARLO achieves multi-level data layout optimization (LO) by changing the linearization of multi-dimensional scientific datasets on 1-dimensional storage space. The multi-level approach combines multiple optimizations to improve the data locality for heterogeneous query access patterns. Unlike traditional post-processing approaches, PARLO applies LO at application run time, before the data is written to storage. By performing in-memory transformation of user data, PARLO is able to eliminate the I/O overhead associated with post-processing.

Figure 3 gives an overview of PARLO as integrated with ADIOS. The core PARLO code is contained in an external library outside of ADIOS, with a minimally-intrusive integration layer within ADIOS that applies PARLO during both write and read operations. On the write side, the integration layer intercepts user I/O operations, invokes the main PARLO code to optimize the buffer layout, and then writes the transformed buffer using standard ADIOS I/O routines. On the read side, the integration layer captures the user's query constraints, passes the constraints to core PARLO to generate read requests for the standard ADIOS read routines, and finally calls PARLO again to process the data from storage to answer the original query.

It is important to note that, although the core PARLO code is cleanly separated from the ADIOS integration layer, from the user's perspective, PARLO operates entirely within the ADIOS workflow. This is because the layout optimization is performed transparently during ADIOS write/read calls. This design achieves transparency in user experience, as well as good software engineering modularity.

Within this framework, PARLO applies a flexible, multi-level layout scheme that combines multiple LO techniques in hierarchical orders, and thus achieves optimization for heterogeneous access patterns. PARLO accepts an ordered list of access pattern requirements, and applies one specific optimization technique for each requirement as a level in the hierarchy. The techniques we implement are: (a) chunking and Hilbert-curve mapping to support queries with spatial constraints (*SC*); (b) value-based binning to support queries with value constraints (*VC*); and (c) byte-level division to support precision-based multi-resolution access. In the following sections, we will examine the PARLO design and implementation in more detail.

B. Run-time Layout Optimization with ADIOS Parallel I/O Middleware

As part of our implementation, we have integrated PARLO with the ADIOS I/O middleware. The user can activate PARLO by adding a simple parameter to the ADIOS XML configuration file, as shown in Figure 4. The value of this parameter ("`transform`") consists of three parts, separated by

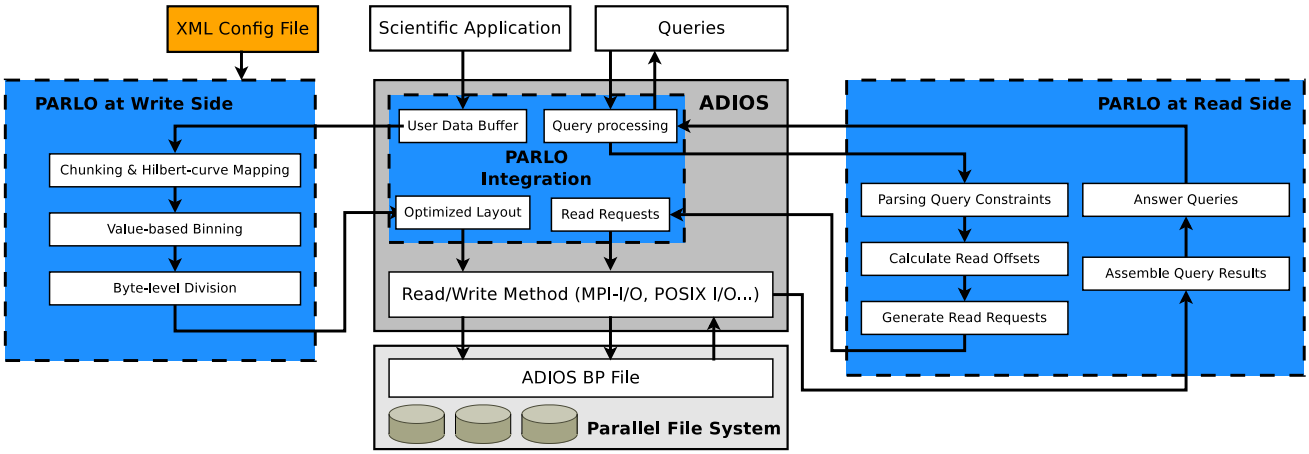


Fig. 3. An overview of PARLO integrated with ADIOS at both write and read side.

```
...
<var name="var1" gwrite="array1"
gread="array1" path="/var" type="double"
dimensions="1,ndx,ndy,ndz"
transform="PARLO:V-M-S|20|8^3"/>
...
```

Fig. 4. ADIOS XML configuration file with PARLO configured.

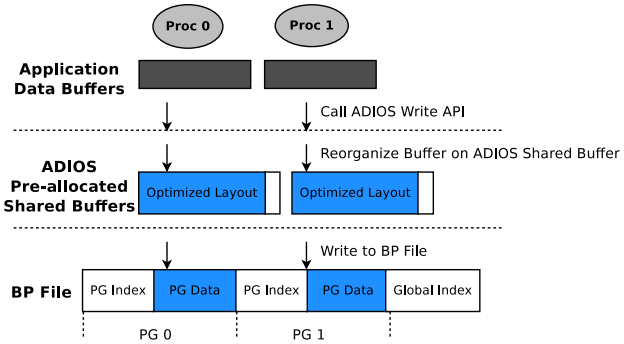


Fig. 5. Layout optimization inside ADIOS.

vertical bars: (1) *PARLO:V-M-S* specifies the LO techniques (V-M-S) PARLO will use to perform the transformation, in descending priority order; (2) *20* indicates the number of bins used for value binning; and (3) *8^3* specifies the spatial chunking parameters (in this case, we specify 8 chunks in each dimension for a 3D dataset). This configuration method allows the user to enable, disable, or reconfigure PARLO easily, without the need to modify or recompile the application code.

In order to further describe our integration, we must first review the internal structure of ADIOS. ADIOS is divided into three main layers: the common layer, the read transport layer, and the write transport layer. The read and write layers are modular, permitting different implementations to be swapped via the XML configuration without recompilation. These layers are both tied to the central common layer via a set of "transport method hooks" (an API of functions that the read/write transport developer must implement).

Given this structure, we aim to restrict all code modifications for PARLO integration to the common layer only. In this way, we ensure that PARLO will be compatible with all current and future read and write transport methods (assuming they are fully conformant to the hook API), and thus will benefit from future low-level I/O optimizations. We have fully achieved this goal with respect to the write transport layer, requiring no modification to write methods whatsoever. The read layer was more difficult, as it does not propagate the new PARLO metadata for the common layer, and because all spatial selection functionality is implemented exclusively within that layer. Thus, some changes were required, but we have minimized the impact by encapsulating the required extensions in new hook API functions, which, for the basic ADIOS "read bp" method, were trivial to implement (as they simply expose already-available metadata).

We achieve this level of separation by transparently replacing user variables with special one-dimensional byte array variables as shown in Figure 5. This allows the read and write layers to operate exactly as before (as if the user had defined these 1D byte arrays explicitly), but gives our integration code the flexibility to perform LO at the byte level. The original metadata for these variables is retained separately, and used to perform translation between this internal representation and the user's variable schema. Whenever data is read or written, PARLO intercepts the user's request, translates it to requests on the corresponding 1D byte array (optimized via our layout methods), delegates to the read/write layer to service these translated requests, and finally performs translation back to the user's view of the variable before returning. This translation is kept completely internal to the common layer: the user sees the variables exactly as they were without PARLO, and the read/write layers see the variables as 1D byte arrays.

Having presented the integration framework, we now describe how our layout optimizations are actually applied within the context of ADIOS. As introduced in Section II-C and

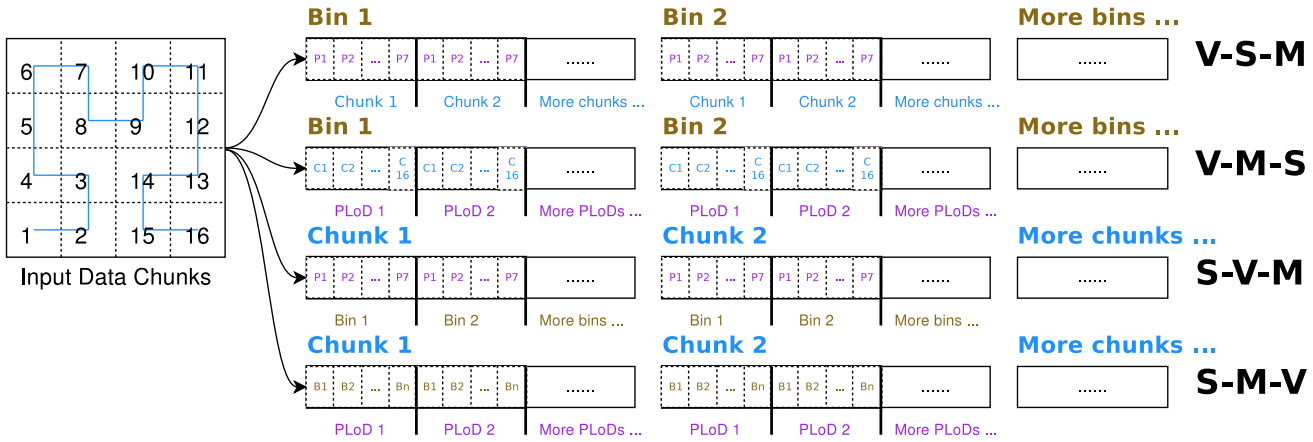


Fig. 6. Example storage layout schemes for intra-PG layout optimization.

Figure 2, the ADIOS BP file format consists of multiple process groups (PGs), each corresponding to the data written by a single process during a single write operation. Our LO technique performs both *intra-* and *inter*-PG optimization; that is, PARLO not only optimizes the data layout within a single PG, but also optimizes the layout of the PGs themselves within the BP file. These two optimization schemes are applied selectively based on application write patterns and the user's requirements for access pattern optimization in order to achieve the most efficient LO.

C. Layout Optimization and Query Index Building

1) Multi-level Intra-PG and Inter-PG Data Layout Optimization: PARLO's intra-PG optimization is based on our previous work, MLOC [10]. It optimizes for heterogeneous access patterns by applying one or more of the following LOs (1) chunking and Hilbert space-filling curve (*HSFC*) mapping, which optimize for spatial-constrained access patterns by improving spatial locality; (2) value-based binning, which optimizes for value-constrained access patterns by grouping data based on values; and (3) byte-level precision-based level of details (APLOD) [11], which offers a precision-based multi-resolution data storage and access solution by breaking value into byte-level components. As shown in Figure 1, the adjustable LO's order placement gives user flexibility of changing optimized data layout according to the priorities of queries in analytic jobs.

Figure 6 shows four examples of how PARLO optimizes for heterogeneous access patterns using different priority orders. Taking the *V-S-M* layout scheme as a specific example, PARLO optimizes for value-constrained access patterns (*V*) at the first level by dividing the whole dataset into multiple bins. Next, within each bin, data is partitioned into chunks, and the chunks are stored in *HSFC* order to optimize for spatial-constrained access patterns (*S*). Finally, within each chunk, the data is further partitioned into multiple APLOD levels to optimize for multi-resolution access patterns (*M*). Similarly, the *S-M-V* layout scheme optimizes for spatial-constrained access patterns as first priority, multi-resolution as second priority, and value-constrained as third priority.

Based on the size of data within a PG, it may be desirable to apply only some of the LO optimizations. For example, if the data size within a PG is small (e.g., several MBs), it may not be necessary to perform chunking to further divide it into smaller chunks. In this case, the user may wish to only apply binning and/or APLOD division to the data. PARLO does support such partial optimization based on data characteristics and user requirements.

Finally, in the case that spatial access (S) is selected for first priority optimization, PARLO also supports inter-PG layout optimization to improve data locality among multiple PGs. Using global communications to reorganize data among PGs can be expensive, and is therefore undesirable. For this reason, PARLO applies communication-free inter-PG LO, in which each process changes its write offset by via a *HSFC* mapping. This requires no inter-PG communication, since each process only needs to know its own offset to perform this calculation. Within each PG, finer-grained LO such as binning and APLOD division can be performed to further optimize for heterogeneous access patterns.

2) Run-time Layout Optimization: The original user data buffer for each PG is transformed in memory at run time to generate the final layout schemes shown in Figure 6, after which ADIOS writes the data buffer with optimized layout to storage. The run-time transformation brings up challenges due to limited run-time resources, including CPU and memory. Extra CPU cycles are required to transform the data into the optimized layout, and extra memory space is required to hold the transformed data. The run-time resource consumption must be controlled in order not to affect the run-time performance of running applications. PARLO addresses the challenge by adopting a lightweight implementation that leverages the ADIOS shared memory buffer to reduce memory footprint. ADIOS supports a "shared memory" writing mode, in which a single buffer is maintained and all the user's *write* calls copy data from user buffer to this shared buffer, with the final *close* call triggering the transfer of the shared buffer to storage via a single I/O operation. As shown in Figure 5,

PARLO directly transforms the user's data buffer into the ADIOS shared buffer, mimicing the original data movement such that no extra intermediate memory is required. As for the CPU time required by buffer transformation, this is shown to be small relative to the existing I/O time in Section IV.

It is important to note that the overhead of LO when applied at run time is, in itself, unavoidable, as the LO would need to be applied at some point anyway. Compared to the traditional post-processing approach, where LO is performed on data already written to disk, PARLO is able to save I/O time, storage space (since in post-processing, both the original and the copy constructed with LO should be stored until processing is complete), and CPU resources, as demonstrated in Section IV.

3) Run-time Value-based Binning: Our previous work [10] applies equal-frequency binning to achieve balanced performance when accessing different bins. This approach works well in post-processing since the bin boundaries can be easily sampled from existing datasets. However, it is challenging to perform binning at run time because the data values are not known beforehand. PARLO offers three possible approaches to achieve efficient run-time binning without inter-process communication, which provide a range of trade-offs to the user: (1) Pre-defined Binning specified by the user. In this case, bin boundaries are given and do not require calculation by PARLO, which saves run-time resources. The limitation of this approach is that it relies on users' knowledge about the value distribution of the datasets to be produced, which is not always feasible. (2) Run-time Equal-Width Binning. In this case, the value range (max/min) is calculated for each PG with a single scan. Then, binning boundaries are determined by dividing the range into a number of equal-size intervals. This method releases PARLO from its dependency on users' knowledge, and has $O(n)$ time complexity, which is an acceptable computational overhead. The drawback is that data with a skewed distribution may not be evenly distributed under this scheme. (3) Run-time Equal-Frequency Binning. This approach first sorts the data within each PG, and sets bin boundaries such that all bins have a (roughly) equal number of elements. This approach achieves best distribution, but requires $O(nlogn)$ time complexity, which may not be suitable for use with large PGs.

It is important to note that binning induces an additional overhead in the extra time and storage space required to write the associated index to disk, which requires 50% of the original size of the raw data when uncompressed. In order to reduce this overhead, we apply Zlib compression to the indexes. Although the trade-off is more CPU cycles spent in compression, the reduction of index size significantly reduces I/O time, and the overall throughput is improved. Since binning is the major source of run-time overhead, PARLO classifies optimization schemes based on run-time overhead. Currently, schemes containing "V" (binning) are tagged as high-overhead, whereas those without "V" are tagged as low-overhead. PARLO is able to make a smart selection among LO schemes based on the user's requirement on run-time performance. When the user requests a high-overhead layout scheme, PARLO estimates whether the schemes can meet the given performance requirement, and if not, PARLO will choose a similar scheme with lower overhead that is able to meet the requirement.

D. Query Support at ADIOS Read Side

PARLO provides rich query functions which are integrated with ADIOS read APIs. Users can specify spatial constraints, value constraints and precision requirements through the extended read APIs to answer queries over the stored data. Queries are integrated with ADIOS by introducing a minimal set of additional query APIs on top of the ADIOS framework.

When handling query constraints, PARLO produces a set of seek-and-read pairs (i.e., byte segment reads) to be passed to the ADIOS read layer to fetch the required data from storage. First, whenever a spatial selection is posed, we intersect it with each PG's bounding box to create a set of smaller, PG-local selections. Then, for each such PG, value and precision constraints are run through PARLO to determine the byte segment reads corresponding to the bins and/or byte-level precision offsets to read from that PG on disk. Finally, all byte segment reads are collected and executed at once by the ADIOS read layer. Since PARLO improves data locality by its storage reorganization, these reads are kept more contiguous on disk than with an unoptimized layout, improving I/O throughput. Moreover, these byte segment reads may be automatically optimized by collective MPI-I/O [22], further reducing I/O overhead.

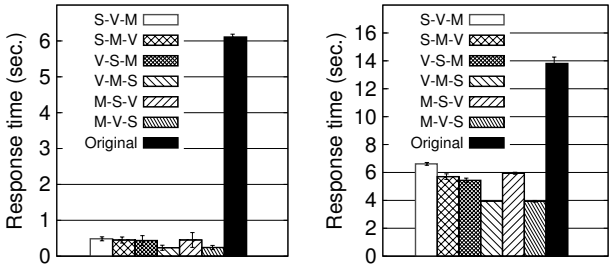
PARLO supports parallel data access through ADIOS to speed up queries. The underlying parallel access is typically achieved via MPI and MPI-I/O. Given an access pattern with a set of constraints, PARLO calculates the bins, blocks and partial bytes to access, and then distributes the data access evenly among all processes. Each MPI process fetches and processes a subset of blocks across the bins. In general, I/O is difficult to balance on parallel file systems, as they are shared by many users with highly dynamic access patterns. Thus, PARLO aims to assign an equal amount of data I/O to each process.

IV. Results

A. Experiment Setup

In order to evaluate PARLO's performance, we conducted experiments on the Lens cluster at Oak Ridge National Lab. Lens is a 77 node Linux cluster dedicated to data analysis and high-end visualization, and is connected to the main Lustre parallel file system at ORNL. Each node is installed with four quad-core 2.3 GHz AMD Opteron processors.

Since PARLO focuses on run-time data layout optimization, we evaluate the performance of PARLO using an I/O simulator for a real scientific application. Specifically, we use the I/O simulator for *Pixie3D* [4], a 3-dimensional extended MHD code that solves the extended MHD equations in 3D arbitrary geometries. The I/O simulator performs the same I/O operations as the original application, calling the ADIOS



(a) 1% dimension selectivity, 5% value selectivity, 2-byte APLOD level.

(b) 10% dimension selectivity, 10% value selectivity, 4-byte APLOD level.

Fig. 7. Query performance of 3-level LO schemes in comparison to queries on the original ADIOS BP file without optimization.

I/O APIs to write data to storage in the BP format. The only difference is that random values are written to the datasets, since the computational part of the simulator is omitted. We compile this I/O simulator with PARLO-integrated ADIOS and compare the run-time performance with and without PARLO enabled to study the run-time overhead introduced.

B. Query Performance Evaluation

We first show the query performance of PARLO during ADIOS read by comparing the query response times with and without using the PARLO optimized layout. Figure 7 shows the query performance comparison for queries with multiple constraints and various selectivity combinations. For PARLO-based schemes, fine-grained 3-level LO are applied in different priority orders to optimize for all access patterns. An 8 GB dataset ($1024 \times 1024 \times 1024$ 3-dimensional double-precision variable) is used for query evaluation. Chunk size is set to $64 \times 64 \times 64$ and bin count is set to 20. Queries are generated with fixed selectivity but random select ranges for the evaluation.

Both figures show significant performance improvements compared to queries on original BP files. For higher selectivities, shown in Figure 7(a), the improvement is 12 to 26 times. For lower selectivities, shown in Figure 7(b), the improvement is 2 to 3.5 times. Thus, under lower selectivities, the layout optimization becomes less effective, since larger portions of data are accessed. This is expected, as it is a well-known fact in the database community that, when accessing a large portion of data in a database (e.g., 1%-10%), sequential read and filtering is comparable to, or even faster than, querying using an index [8]. Therefore, when overall selectivity is greater than 1%, PARLO falls back to sequential read and filtering to ensure acceptable query performance. Our previous work [10] contains additional, detailed results and analysis of query performance, along with comparisons to existing scientific data management systems.

C. Run-time Layout Optimization Performance Evaluation

Next, we evaluate the run-time overhead of PARLO. We do this by running the *Pixie3D* I/O simulator for one timestep, which produces a 3-dimensional double-precision floating-point dataset. We use 64 processes to write the datasets, with each process writing 8 variables. The domain of each process is $256 \times 256 \times 256$, yielding a data size per process of 1 GB, and a total size of 64 GB. When applying PARLO, for schemes optimized for value-constrained access patterns (i.e., those containing "V"), the bin count is set to 20, and equal-width dynamic binning is used to calculate bin boundaries at run time. Zlib compression at compression level 1 is applied on the binning indexes to reduce data size and I/O time. Higher compression levels were tried but proved ineffective in further reducing the index size, yet increased compression time by over two times.

TABLE I
STORAGE OVERHEAD COMPARISON BETWEEN PARLO AND POST-PROCESSING.

Original	PARLO no "V"	PARLO w/ "V"	Post-proc. no "V"	Post-proc. w/ "V"
64 GB	65 GB	76 GB	129 GB	140 GB

The end-to-end data write time is measured for the *Pixie3D* I/O simulator both under original ADIOS (i.e., no LO), and under all permutations of run-time LO offered by PARLO. The processing time of the traditional post-processing approach is also measured for each of these LO schemes, which consists of the time to: 1) write the data to storage with the original version of ADIOS; 2) apply equal-width binning to get bin boundaries for fair comparison; 3) load the dataset into memory and perform LO; and 4) write layout-optimized data back to storage. The results are shown in Figure 8. All bars in the graph represent the percent overhead of the given method relative to original ADIOS with no LO applied. Compared to post-processing, PARLO reduces processing time by an average of 56%. Furthermore, as shown in Table I, PARLO saves 46% to 50% storage space by writing data with optimized layout directly to storage, avoiding the storage of redundant intermediate data.

Another interesting results from Figure 8 is that, compared to original ADIOS performance, non-binning LO schemes (those without "V") only increase write time by an average of 3.3%, which is very little overhead. For schemes with binning (with "V"), the overhead is somewhat greater due to the binning indexes built at run time. These indexes start at 50% of original data size, though after zlib compression they are reduced by 64%, thus reducing the total write size by 21% (shown in Table I). As a trade-off, an average of 4.5% more CPU time is required due compression. In the end, the write time for LO schemes with binning is on average 30% greater than that of original ADIOS without LO. Considering that this form of binning is equivalent to building and storing a query index at run time, and given the query performance improvement shown above, this overhead seems reasonable.

V. RELATED WORK

Enormous data size and disparity between computing capability and I/O bandwidth cause I/O to be the bottleneck

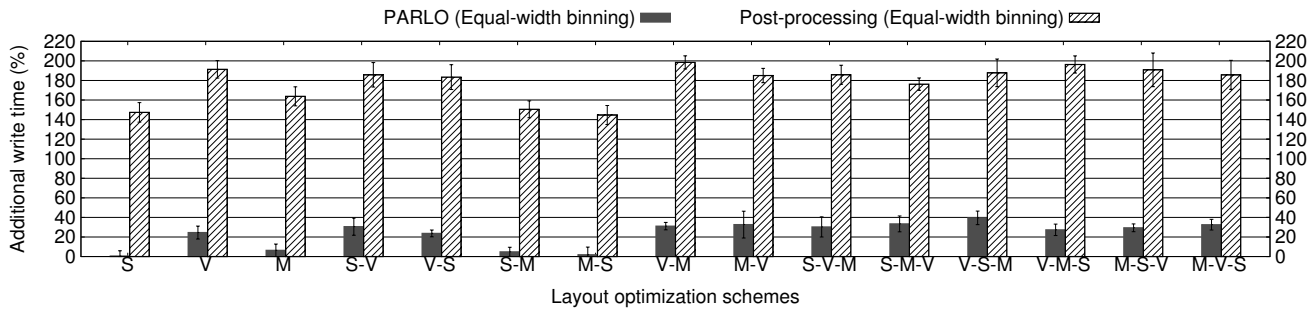


Fig. 8. Additional end-to-end data write time of *Pixie3D* I/O simulator compared to the original performance of ADIOS without any run-time LO.

in modern HPC systems, making exploration and analysis on big data a challenging problem. A collection of data layout optimization methods have been presented to improve data access performance for queries. Data binning approaches with various forms of indexing, including FastBit [25] and ISABELA-QA [16], have been applied to speed up value-constrained access patterns. SciDB [2] optimizes for spatial-constrained access patterns with array slicing and joining, and array division into regular/irregular chunks [21]. The application of space-filling curves [17], for example the Hilbert curve, to improve spatial locality when accessing multi-dimensional scientific datasets has also been explored [12], [15], [23], [24]. However, while these methods have been successful in optimizing for specific query types, they do not address the issue of heterogeneous access patterns.

A dynamic layout scheme with hybrid replications for complex I/O patterns has been developed [20], as well as a method of distributing workload among busy and idle servers based on I/O cost measurement [19]. While successful in improving I/O performance, these methods rely on knowledge of specific application data access patterns. In contrast, PARLO addresses heterogeneous access patterns induced by a range of general query types. Unlike prior post-processing approaches [5], [25], PARLO is integrated with parallel I/O middleware to achieve efficient run-time in-memory layout optimization and index building.

HPC I/O libraries are designed to work as middleware to achieve efficient I/O for applications, and to provide portable, metadata-rich data storage formats. PnetCDF [13], one such popular middleware code, applies linear data layout in its file structure so that arrays are stored either in a contiguous or interleaved way. Another, HDF5 [7], applies a hierarchical group structure that offers flexibility of data layout, including contiguous, chunked, and compact layouts. HDF5 also implements a "filter" pipeline, which allows the user to specify run-time transformation. However, this filter pipeline is only designed for data compression, and cannot be easily extend to incorporate run-time transformations such as the storage layout optimization presented in this paper. Another common I/O middleware, ADIOS [14], has already been described in detail previously. ADIOS supports many data transport methods (including POSIX, MPI-IO [22], and staged reading/writing) and output formats (including the default BP (Binary Packed), pnetCDF, and HDF5). However, in relation to PARLO, it does not yet provide intrinsic support for run-time data transformation, which motivates the ADIOS integration work in this paper.

As an alternative to in situ data processing on the compute cores, methods for moving data to a staging cluster for processing have been explored in some detail. DataStager [1] is a framework that moves data asynchronously from compute nodes to staging nodes, and enables data processing methods such as aggregation, partitioning, and validation to be run in the staging area. PreDatA [26], an extension of DataStager, characterizes and reorganizes application data to speed up data analytics. Though these approaches have many benefits in terms of reducing compute cluster load, speeding up I/O, and enabling new types of *in transit* analytics and processing, they are only applicable when staging resources are available. Because PARLO is integrated with I/O middleware, it can be flexibly deployed at different positions along the data flow. Because of this, PARLO can take advantage of staging cores if available, but is also applicable using compute cores alone.

VI. Conclusion

In this paper, we present PARLO, a parallel run-time layout optimization framework integrated into the ADIOS I/O middleware, to improve the performance of queries with heterogeneous access patterns on multi-dimensional, spatio-temporal scientific datasets. PARLO offers users a flexible, transparent method to achieve run-time data layout optimization without modifying or re-compiling application code, controlled with simple, XML-based configuration. Based on this optimized data layout, PARLO also provides rich query functions integrated into the ADIOS read API to improve the performance of queries with heterogeneous access patterns.

Experiments show that PARLO can improve query performance by 2 to 26 times via run-time data layout optimization and query index build, while still maintaining a reasonable run-time I/O overhead of between 3% and 30%. Compared to traditional post-processing approaches, PARLO reduces the processing time by 56% and storage overhead by up to 50%.

VII. Acknowledgment

We would like to acknowledge the use of resources at ORNL's leadership class computing facility, OLCF. Also, we

appreciate the use of the datasets available from the Flash Center for Computational Science. This work was supported in part by the U.S. Department of Energy, Office of Science and the U.S. National Science Foundation (Expeditions in Computing and EAGER programs). Oak Ridge National Laboratory is managed by UT- Battelle for the LLC U.S. D.O.E. under contract no. DEAC05- 00OR22725.

REFERENCES

[1] H. Abbasi, M. Wolf, G. Eisenhauer, S. Klasky, K. Schwan, and F. Zheng. DataStager: scalable data staging services for petascale applications. In *Proceedings of the 18th ACM Int's Symposium on High Performance Distributed Computing*, HPDC '09, pages 39–48, New York, NY, USA, 2009. ACM.

[2] P. G. Brown. Overview of SciDB: Large scale array storage, processing and analysis. In *Proceedings of the 2010 International Conference on Management of Data*, SIGMOD '10, pages 963–968, New York, NY, USA, 2010. ACM.

[3] P. H. Carns, W. B. Ligon, III, R. B. Ross, and R. Thakur. PVFS: a parallel file system for Linux clusters. In *Proceedings of the 4th annual Linux Showcase & Conference - Volume 4*, pages 28–28, Berkeley, CA, USA, 2000. USENIX Association.

[4] L. Chacn. A non-staggered, conservative, VB=0' finite-volume scheme for 3D implicit extended magnetohydrodynamics in curvilinear geometries. *Computer Physics Communications*, 163:143–171, 2004.

[5] J. Chou, K. Wu, and Prabhat. Fastquery: A parallel indexing system for scientific data. In *Proceedings of the 2011 IEEE International Conference on Cluster Computing*, CLUSTER '11, pages 455–464, Washington, DC, USA, 2011. IEEE Computer Society.

[6] S. Donovan, G. H. Ibm, A. J. Hutton, A. J. Hutton, C. C. Ross, C. C. Ross, L. Symposium, L. Symposium, L. Symposium, M. K. Petersen, W. O. Source, and P. Schwan. Lustre: Building a file system for 1,000-node clusters. In *Proceedings of the Linux Symposium*, 2003.

[7] M. Folk, G. Heber, Q. Koziol, E. Pourmal, and D. Robinson. An overview of the HDF5 technology suite and its applications. In *Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases*, AD '11, pages 36–47, New York, NY, USA, 2011. ACM.

[8] G. Gardin and P. Valduriez. *Relational Databases and Knowledge Bases*. Addison-Wesley, Menlo Park, CA, USA, 1989.

[9] Z. Gong, S. Lakshminarasimhan, J. Jenkins, H. Kolla, S. Ethier, J. Chen, R. Ross, S. Klasky, and N. F. Samatova. Multi-level layout optimization for efficient spatio-temporal queries on ISABELA-compressed data. In *Proceedings of the 26th IEEE International Parallel and Distributed Processing Symposium*, IPDPS '12, 2012.

[10] Z. Gong, T. Rogers, J. Jenkins, H. Kolla, S. Ethier, J. Chen, R. Ross, S. Klasky, and N. F. Samatova. MLOC: Multi-level layout optimization framework for compressed scientific data exploration with heterogeneous access patterns. In *Proceedings of the 41st International Conference on Parallel Processing*, ICPP '12, pages 239–248, 2012.

[11] J. Jenkins, E. Schendel, S. Lakshminarasimhan, T. Rogers, D. A. Boyuka, S. Ethier, R. Ross, S. Klasky, and N. F. Samatova. Byte-precision level of detail processing for variable precision analytics. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC 2012)*, 2012.

[12] J. K. Lawder and P. J. H. King. Querying multi-dimensional data indexed using the Hilbert space-filling curve. *SIGMOD Rec.*, 30:19–24, March 2001.

[13] J. Li, W. keng Liao, A. Choudhary, R. Ross, R. Thakur, W. Gropp, R. Latham, A. Siegel, B. Gallagher, , and M. Zingale. Parallel netCDF: A high-performance scientific I/O interface. In *Proceedings of SC2003*, Nov. 2003.

[14] J. F. Lofstead, S. Klasky, K. Schwan, N. Podhorszki, and C. Jin. Flexible IO and integration for scientific codes through the adaptable IO system(ADIOS). In *Proceedings of the 6th International Workshop on Challenges of Large Applications in Distributed Environments (CLADE '08)*, pages 15–24, 2008.

[15] V. Pascucci. Global static indexing for real-time exploration of very large regular grids. ACM Press, 2001.

[16] S. Lakshminarasimhan, J. Jenkins, I. Arkatkar, Z. Gong, H. Kolla, S-H Ku, S. Ethier, J. Chen, C.S. Chang, S. Klasky, R. Latham, R. Ross and N. F. Samatova. ISABELA-QA: Query-driven data analytics over ISABELA-compressed scientific data. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC 2011)*, Seattle, Washington, 2011.

[17] H. Sagan. *Space-Filling Curves*. Springer-Verlag, New York, NY.

[18] F. Schmuck and R. Haskin. GPFS: A shared-disk file system for large computing clusters. In *First USENIX Conference on File and Storage Technologies (FAST'02)*, Monterey, CA, 2002.

[19] H. Song, H. Jin, J. He, X.-H. Sun, and R. Thakur. A server-level adaptive data layout strategy for parallel file systems. In *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2012 IEEE 26th International*, pages 2095 –2103, may 2012.

[20] H. Song, Y. Yin, Y. Chen, and X.-H. Sun. A cost-intelligent application-specific data layout scheme for parallel file systems. In *Proceedings of the 20th international symposium on High performance distributed computing*, HPDC '11, pages 37–48, New York, NY, USA, 2011. ACM.

[21] E. Soroush, M. Balazinska, and D. Wang. ArrayStore: A storage manager for complex parallel array processing. In *Proceedings of the 2011 International Conference on Management of Data*, SIGMOD '11, New York, NY, USA, 2010. ACM.

[22] R. Thakur, W. Gropp, and E. Lusk. Data sieving and collective I/O in ROMIO. In *Proceedings of the The 7th Symposium on the Frontiers of Massively Parallel Computation*, FRONTIERS '99, pages 182–, Washington, DC, USA, 1999. IEEE Computer Society.

[23] Y. Tian, S. Klasky, H. Abbasi, J. Lofstead, R. Grout, N. Podhorszki, Q. Liu, and Y. W. W. Yu. EDO: Improving read performance for scientific applications through elastic data organization. In *Proceedings of the 2011 IEEE International Conference on Cluster Computing*, Cluster '11, Austin, TX, USA, 2011. IEEE.

[24] Y. Tian, S. Klasky, W. Yu, H. Abbasi, B. Wang, N. Podhorszki, R. Grout, and M. Wolf. SMART-IO: System-aware two-level data organization for efficient scientific analytics. In *Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS), 2012 IEEE 20th International Symposium on*, pages 181–188, Aug.

[25] K. Wu. FastBit: An efficient indexing technology for accelerating data-intensive science. *Journal of Physics: Conference Series*, 16(1):556, 2005.

[26] F. Zheng, H. Abbasi, C. Docan, J. Lofstead, S. Klasky, Q. Liu, Manish-Parashar, N. Podhorszki, K. Schwan, and M. Wolf. PreDatA–preparatory data analytics on peta-scale machines. In *Proceedings of the 26th IEEE International Parallel and Distributed Processing Symposium*, IPDPS '10, Atlanta, GA, April 2010.