EDO: Improving Read Performance for Scientific Applications Through Elastic Data Organization

Yuan Tian[1] Scott Klasky[2] Hasan Abbasi[2] Jay Lofstead[3] Ray Grout[4]
Norbert Podhorszki[2] Qing Liu[2] Yandong Wang[1] Weikuan Yu[1]

Auburn University[1]
{tianyua,yzw0025,wkyu}@auburn.edu

Oak Ridge National Laboratory[2]
{klasky,habassi,liuq,pnorbert}@ornl.gov

Sandia National Laboratories[3]
{gflofst}@sandia.gov

National Renewable Energy Laboratory[4]
{Ray.Grout}@nrel.gov

***Abstract*—Large scale scientific applications are often bottlenecked due to the writing of checkpoint-restart data. Much work has been focused on improving their write performance. With the mounting needs of scientific discovery from these datasets, it is also important to provide good read performance for many common access patterns, which requires effective data organization. To address this issue, we introduce Elastic Data Organization (EDO), which can transparently enable different data organization strategies for scientific applications. Through its flexible data ordering algorithms, EDO *harmonizes* different access patterns with the underlying file system. Two levels of data ordering are introduced in EDO. One works at the level of data groups (a.k.a process groups). It uses Hilbert Space Filling Curves (SFC) to balance the distribution of data groups across storage targets. Another governs the ordering of data elements within a data group. It divides a data group into subchunks and strikes a good balance between the size of subchunks and the number of seek operations. Our experimental results demonstrate that EDO is able to achieve balanced data distribution across all dimensions and improve the read performance of multidimensional datasets in scientific applications.**

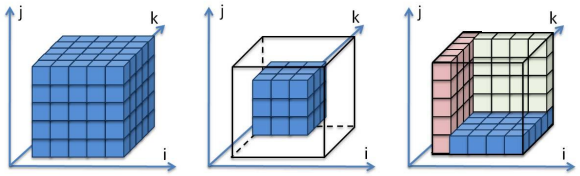
***Keywords*-Data Organization; Space Filling Curve; ADIOS; Parallel I/O; Planar Read Patterns.**

I. Introduction

Large-scale simulation codes can generate massive multidimensional datasets, from checkpoint-restarts, monitoring, analysis, and visualization output. Their execution is often bottlenecked by the cost of I/O because of their gigantic datasets. Many efforts have focused on decreasing the application turnaround time by studying the output side of the problem, but few have systematically examined the read performance of scientific applications on large-scale supercomputers, despite the importance of read performance to scientific simulation and analysis workflows.

To improve read performance, a thorough understanding of application access patterns is crucial. Based on the authors' direct experience with many application teams in the U.S. and beyond, including combustion (S3D [6]), fusion (GTC [20], GTS [45], XGC-1 [5]), earthquake simulation (SCEC [10]), MHD (pixie3D [4]), numerical relativity codes (PAMR [37]), and supernova (Chimera [29]) codes, there are four main fundamental reading patterns for application data analysis:

- Read all of a single variable (c.f. Figure 1(a)). This would be representative of reading the temperature across a simulation space, for example.
- Read an arbitrary orthogonal subvolume (c.f. Figure 1(b)).
- Read an arbitrary orthogonal full plane (c.f. Figure 1(c)).
- Read multiple variables together. This would be representative of reading the components of a magnetic field vector, for example.



(a) Variable (b) Subvolume (c) 2-D Planes

Fig. 1: A $5 \times 5 \times 5$ Array (k: fastest dimension)

Other reading patterns are either composed of a mixture of these patterns or are minor variations. For example, reading entire checkpoint-restart datasets can be perceived as an extended case of reading multiple variables together. Figure 1 gives an example of a $5 \times 5 \times 5$ 3-D array (Figure 1(a)), a $3 \times 3 \times 3$ subset (Figure 1(b)) of the array and three 5×5 planes (Figure 1(c)) in three dimensions: i, j, and k; where i is the primary, i.e., the slowest varying dimension, and k is the tertiary and fastest varying dimension. Data in the array is stored first along the fastest dimension k, then along the slow dimensions, j and i, on disk.

Among these patterns, reading orthogonal planes has been the least studied. However, it is a very commonly used data pattern by scientific applications. For example, for combustion studies with S3D [16], the computation was targeted at a variable of $1408 \times 1080 \times 1100$ points (12GB), but the majority of analysis is performed on the orthogonal planes of the variable (either 1408×1080 or 1080×1100 points). However, the performance of reading such planes (a.k.a *planar read*)

978-0-7695-4516-5/11 $26.00 © 2011 IEEE
DOI 10.1109/CLUSTER.2011.18

is often bottlenecked by the extremely poor performance in retrieving data along slow dimensions from multidimensional arrays [8].

There are two main issues faced by such access patterns of multidimensional arrays. The first is the disparity between the order of storage and the order of access for data elements in a multidimensional variable. When data is not traversed in the order in which it is stored, reading cannot benefit from techniques such as data prefetching, caching, etc. For example, when reading an *ij* plane, data elements are stored non-contiguously along these slow-varying dimensions. This requires a large number of time-consuming seek and read operations, so that the read performance degrades significantly. However, this does not happen to *jk* planes whose data elements are stored along the fast-varying dimensions. Such disparity leads to a phenomenon called *performance bias* against the slow-varying dimensions. A current popular solution to this problem is to store multiple copies of the same data with a different dimension being used as the primary dimension in each copy. For example, climate researchers at the Geophysical Fluid Dynamics Laboratory (GFDL) make multiple replicas of all datasets with *x*, *y*, *z* and *time* as the fastest dimension, respectively. Such workarounds help reduce the reading time [11], but increase the total storage size by 4 times. Second, there is a lack of data concurrency when a subset of data elements is retrieved from a multidimensional variable. When the subset of data elements is not logically contiguous, it is often concentrated on only a few storage devices among a large number of total devices that are used to store the entire variable. For example, with the LC layout, a plane in the fast dimension will be located on only one storage target if the plane size is less than the stripe size. In this case, applications can not make use of aggregated bandwidth from all devices, and are then limited to the bandwidth available from a single storage device. Theses issues are examined in detail in Section II-A.

To enable fast access of multidimensional scientific datasets, it is important to investigate a strategy that can address both issues. In this paper, we propose an I/O framework named Elastic Data Organization (EDO) that can support flexible data organization strategies for different scientific applications. EDO addresses the challenging issues faced by planar reads through two levels of data ordering algorithms. At the top level, it uses Hilbert Space Filling Curves (SFC) to balance the distribution of data groups (a.k.a process groups) across storage targets. As we will discuss in Section III, SFC distributes data elements across parallel storage targets to aggregate their bandwidth without file system restrictions and improve concurrency for the aforementioned access patterns. At a low level, EDO divides data elements within a data group into subchunks, and balances the cost of seeking through subchunks and that of reading data from them. Neither ordering algorithm precludes portability nor requires any application-level changes. They are used to decide the placement of data elements for optimal concurrency and better exploitation of bandwidth from storage devices.

We evaluate the performance improvement for planar reads on the Jaguar supercomputer at Oak Ridge National Laboratory. We show that the performance of planar reads can be improved by EDO in a balanced manner across all dimensions. A maximum speedup of 37 times has been observed.

The rest of the paper is organized as follows. In Section II, we introduce the background for this work. We then describe the design of EDO in Section III. Section IV provides a mathematical analysis of data concurrency using different data organization strategies. Section V further validates our strategy through a comprehensive set of experimental results. Section VI provides an overview of related work. Finally, we conclude the paper in Section VII.

II. Background

In this section, we present a short discussion of existing data organizations and their performance issues. EDO is derived from ADIOS (Adaptable I/O System), an I/O middleware from Oak Ridge National Laboratory. We also provide an overview of ADIOS and its BP (Binary Packed) file format.

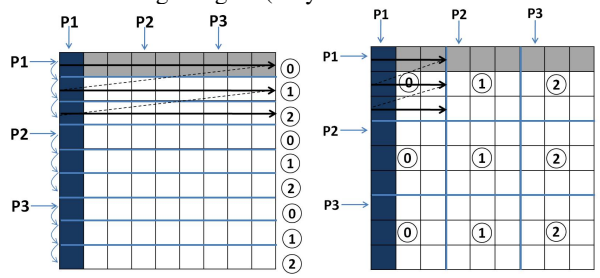
A. Common Data Organizations

When an application needs to retrieve data elements from a multidimensional dataset, two main factors affect the read performance. One is the contiguity of these data elements, another is the number of concurrent storage devices that are supplying the data. The former determines the maximum number of seek operations, though the actual number of seeks may be reduced by reading extra data between data elements. The latter determines the *concurrency* of storage access in an application.

Currently there are two popular data organizations: logically contiguous (LC) and chunking. Figure 2 compares these two data organizations and show how the read performance can be different between these organizations. In the figure, a 2-D array with 9×9 integer elements is written on three storage targets using LC and chunking, respectively. The stripe width is equal to 36 bytes. The arrowed lines represent the order in which these data elements are stored on storage devices, e.g., Object Storage Targets (OSTs) in the case of Lustre file system. The circled numbers indicate targets on which data elements are located; the shaded squares are the requested data elements. If we read in the row-major order with three processes, for both organizations, each process needs 1 seek operation and 1 read operation to retrieve the data. However, chunking is expected to be 3 times faster than LC since LC serializes read requests from three processes to one OST.

Concurrency issues are also observed for LC when retrieving a column, as shown in Figure 2. Furthermore, the performance is degraded because each process either has to perform 3 seeks and 3 reads to retrieve the data, or one read to get 19 elements at a time. The former is slow due to frequent expensive seek calls; the latter is also inefficient since 84% extra data is retrieved. Chunking can not help for such retrieval patterns as well. Every process either needs the same number of seek and read operations to obtain the requested data, or

retrieves 67% extra data. Meanwhile, chunking suffers from similar concurrency issues. Processes are contending at a small number of storage targets (only *OST0* in this case).



(a) LC (b) Chunking

Fig. 2: Current Data Organizations

B. ADIOS and BP File Format

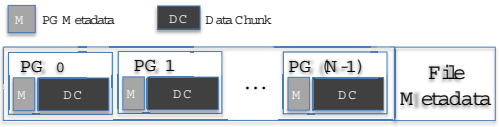


Fig. 3: Data Organization of BP File Format

ADIOS has demonstrated significant performance benefits for a number of petascale scientific applications [1], [25], [27], [47], [36]. It uses a default file format called BP. In this format, ADIOS applies the chunking strategy for storing multidimensional datasets. Each process is assigned one data chunk from one dataset after domain decomposition. If the application needs to output multiple datasets, each process will have multiple data chunks, one from each dataset. A group of chunks from one process, along with their attributes such as data size and offsets, are grouped together and stored as a larger unit, called *Process Group* (PG) in ADIOS. We also refer to it as a *Data Group*. An example of BP file output for one dataset written by N processes is shown in Figure 3. All PGs are placed within BP file in the order of process IDs.

III. Design of Elastic Data Organization (EDO)

In view of the performance issues of existing data organizations, we design EDO as an extension of ADIOS to support elastic data organization algorithms. EDO retains many salient features of ADIOS, including NSSI [34] for staging, DataTap [2] for asynchronous I/O, and Dataspace [12] for memory-to-memory code coupling. Figure 4 shows the software architecture of EDO. It focuses on enabling elastic organization algorithms for different scientific applications. EDO supports *Multi-level Data Organization*. Varying strategies are provided as selectable algorithms to determine the placement of data units in EDO. These include the default linear placement, Hilbert Space Filling Curve (SFC), and subchunking. Z-curve ordering is still in the development. In the rest of the section we discuss two algorithms Hilbert SFC and subchunking in detail, and describe how they are used to determine different levels of data organization in EDO.

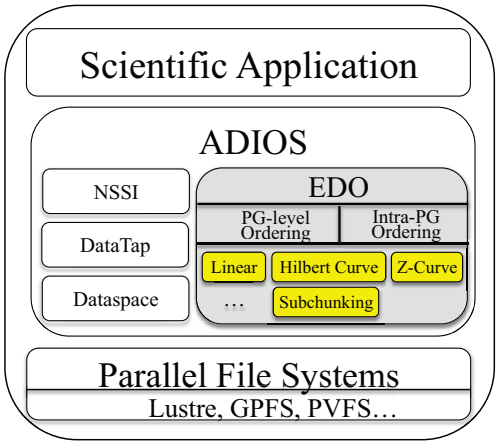


Fig. 4: EDO Architecture

A. Multi-Level Data Organization in EDO

EDO formulates the ordering of data placement into two levels. At a top level, it uses Hilbert Space Filling Curves (SFC) to balance the distribution of data groups (i.e., PGs) across storage targets. At the low level, EDO divides data elements within a data group into subchunks, and organizes data in subchunks.

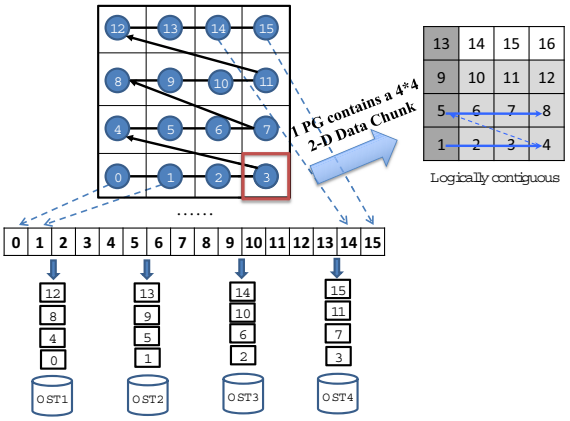
Figures 5(a) and 5(b) compare data organization between the linear placement (as in ADIOS) and the Hilbert SFC-based placement as introduced in EDO. A 2-D array of 16 chunks is used here as an example to simplify the description. These chunks are written to 4 storage targets (OSTs) via 16 processes.

In the original ADIOS, each PG is placed on one storage device in a round-robin fashion. Good concurrency can be achieved in row-major order because sequential PGs are placed on different OSTs, leading to good data distribution. However, such placement may face severe concurrency issues when data is accessed along the slow-varying dimensions, similar to the case in Figure 2(b), .

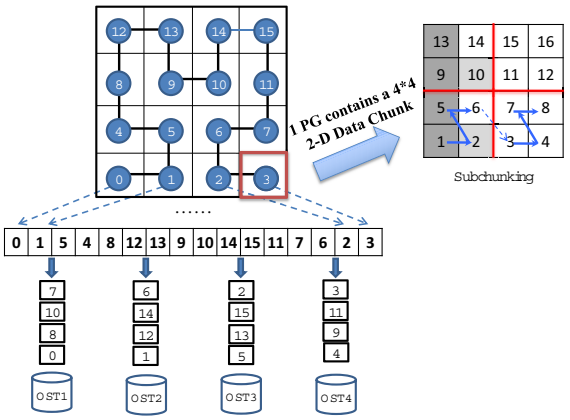
A better method is needed to order PGs so that data chunks on any dimension can be clustered, i.e., achieve good data locality. We use the Hilbert *Space Filling Curve* (SFC) [17] because it is a strategy to map a multidimensional space onto a one-dimensional space, and has been used in a wide variety of applications, especially when data locality is of concern. The Hilbert curve guarantees the best geometric locality properties [33]. Additionally, the cost of transforming the index of data units is low [19], a strength also shared by other strategies such as the morten (z-curve) index.

The 2-D array on the left of Figure 5(b) shows the placement of PGs using Hilbert curve. Using this algorithm, data chunks are shuffled among all OSTs. For example, instead of writing to OST2, chunk 5 will be placed on OST3, while chunk 14 will be placed on OST2 instead of OST3. This strategy does not impact the read performance of the entire array because the number of total storage targets remains the same.

A slight impact is observed for the row-major order, such as the first and second rows, where data chunks are now placed on fewer storage targets. However, a significant difference is shown for data access in the column-major. Instead of striding among multiple rows, many data chunks become sequentially organized. Thus when one column is requested, the targeted data chunks will spread to 3 OSTs, compared to only 1 OST under the original data organization. Thus the Hilbert curve can improve concurrency for data planes from slow-varying dimensions.



(a) Linear Placement in ADIOS



(b) SFC-based Placement with EDO

Fig. 5: Comparison of Different Data Organizations

EDO supports a second level of data organization: intra-PG ordering. This level governs the order of data elements inside a PG. For applications that generate gigantic datasets, their PGs can grow very large to the extent in which data elements inside a PG are sparsely retrieved for certain visualization and the analytics tasks. As discussed in section II-A, one way to read in column-major order with chunking involves retrieving redundant data. Such approach is efficient when data chunks are small, where sequential read has less overhead than frequent seek operations. However, when data chunks are large, the time of retrieving extra data can be significant. For example, retrieving a column from a 2-D array with 1200*1200 elements requires 92% read overhead.

Thus, we introduce a subchunking algorithm at the intra-PG level. Subchunking divides data in a PG into smaller subchunks and order them according to the chunking strategy. As shown in Figure 5, one PG contains one data chunk; each chunk contains many data elements, 16 in this case. In the original ADIOS, such elements inside a data chunk are ordered in a logically contiguous manner. With subchunking, a chunk is divided into 4 *subchunks*. These subchunks are then stored based on the linear ordering. Note that, at present, SFC-based ordering is not necessary inside an ADIOS chunk, because the number of subchunks is typically small. EDO does allow other ordering algorithms when the need arises. With such organization, the overhead of retrieving on column-major can be decreased significantly. As show in Figure 5, the overhead is reduced from 9 to 2 elements (light shaded blocks) when the column (dark shaded blocks) is requested, with 1 more seek operation required.

IV. Analytical Modeling of Data Concurrency

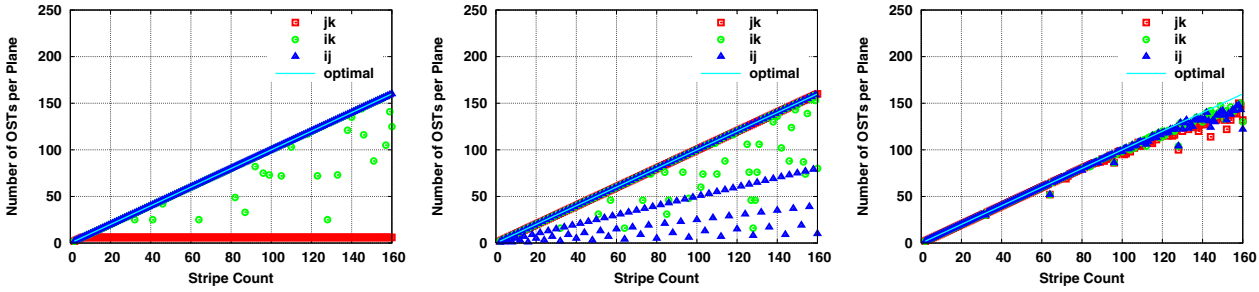
To validate the function of new organization algorithms in EDO, we analytically model the performance of different data organizations, where a 2-D plane is retrieved from a 3-D dataset. We first introduce one formula to quantify the data concurrency. Our study is based on the Lustre [9] file system. The concurrency of a plane is determined by the number of OSTs that data elements are placed. While the placement of data elements are determined by their offsets within the file and striping parameters. Thus, we introduce the following formula to calculate the concurrency:

$$Concurrency = |\alpha_0 \bigcup \alpha_1 \bigcup ... \bigcup \alpha_i|, where$$
$$\alpha_i = \frac{offset_i}{stripe\_width} \% stripe\_count, i \in [0...(n-1)] \quad (1)$$

where n is the number of data elements on the plane. Note that the offset of data element is different under different data organizations.

Based on the above formula, several programs are developed to calculate the concurrency of reading 2-D planes from a 3-D data array. We then compare the average concurrency across three dimensions. Figure 6 shows the result of concurrency when reading 2-D planes from an $800 \times 800 \times 800$ 3-D global array. The initial array is created by 4,096 processes. Each process writes $50 \times 50 \times 50$ elements either as a 3-D chunk or a contiguous segment, depending on the data organization. The stripe size is 1MB. With such dataset, we vary the stripe count from 2 to 160, the maximum stripe count allowed by Lustre, and compare the data concurrency on three dimensions using different organization strategies. Note that the theoretical maximum concurrency for a plane equals to the configured stripe count for the output file. An optimal data organization should show a concurrency that is equal or close to the maximum concurrency on all dimensions.

Under the logically contiguous organization, shown by Figure 6(a), the average number of OSTs for *jk* planes



(a) Logically Contiguous (b) Linear Placement (c) Hilbert Curve

Fig. 6: Concurrency Modeling of a 2-D Plane on 3 Dimensions

is consistently small. Severe contention will occur if such plane is retrieved by many processes in parallel. The data concurrency of *ik* planes varies. In some cases, It drops to less than half of the stripe count. The ij planes have the highest concurrency. Note that the highest concurrency does not necessarily mean the best performance which also depends on the contiguity of the read operations.

Figure 6(b) shows the data concurrency using the linear placement of data chunks. The *jk* planes have the highest concurrency, i.e., they are able to use the maximum number of OSTs. Due to high variances of concurrency on the *ik* and ij planes, low concurrency for these two planes is common. For example, when a stripe count is set to the maximum 160 on Lustre, where the highest aggregated write bandwidth is expected, the resulting concurrency for planar reads is rather low to 80 and 10 for the *ik* and ij planes, respectively.

Figure 6(c) shows the data concurrency of Hilbert curve. All three types of planes show close-to-optimal concurrency, more than 76% of the available storage targets are utilized. There is little variation among different types of planes. Thus, a balanced performance can be expected. When the stripe count for an array is set to 160, the Hilbert curve organization is able to achieve a concurrency of 132, 130 and 122 OSTs, respectively for *jk*, *ik*, and ij planes. There are cases that the linear placement can utilize more OSTs than the Hilbert curve, for instance when the stripe count is 121. But such sweet spots require either a sophisticated calculation that involves the number of read processes, the stripe count, the stripe size, the size of dataset, and/or an extensive set of tuning experiments. Worse yet, one set of parameters will not fit for different multidimensional data arrays. Application scientists would rather be relieved from having to understand such calculation or go through time-consuming tuning experiments. The Hilbert curve-based ordering frees them from such performance concerns. Application scientists can then intuitively choose the striping parameters for their datasets, and/or expect a consistent and well balanced read performance in return.

V. Experimental Performance Results

We deploy EDO on the *Jaguar* supercomputer at Oak Ridge National Laboratory (ORNL) to evaluate its performance. Jaguar is currently the second fastest supercomputer in the world [31]. It is a massively parallel, distributed memory system composed of a 2.3 PetaFlop/s Cray XT5 partition and a 263 TeraFlop/s Cray XT4 partition, a 5 PB file system known as *Spider*. The Cray XT5 partition is used for our experimental evaluation. It contains 18,688 compute nodes besides login/service nodes. Each compute node contains dual hex-core AMD Opteron 2435 (Istanbul) processors running at 2.6GHz, 16GB of DDR2-800 memory, and a SeaStar 2+ router. The entire partition contains 224,256 processing cores and 300TB of memory. The Spider file system is the largest Lustre file system in the world, with 672 storage targets (OSTs) on its widow-1 section and over 26,000 clients, and it is the fastest Lustre file system in the world with a demonstrated bandwidth of 240 GB/s.

A self-contained I/O kernel for S3D [6] from Sandia National Laboratories is used in our experiments. S3D is a high-fidelity, massively parallel solver for turbulent reacting flows. It employs a 3-D domain decomposition to parallelize the simulation of combustion. S3D generates datasets of different sizes. Four test cases: small (S), medium (M), large (L) and extra large (X) are shown in Table I.

TABLE I: Test Cases Written by 16×16×16 Writers

	Per Process		Entire Array	
	Elements	Data Size	Elements	Data Size
S	20^3	62.5KB	320^3	250MB
M	50^3	0.95MB	800^3	3.8GB
L	100^3	7.6MB	1600^3	30.5GB
X	250^3	119.2MB	4000^3	476.8GB

According to the previous practice with ADIOS on Jaguar, the stripe size is set as the size of PG, a technique that maximizes concurrency and reduces false sharing on the Lustre file system. A separate test program is created to read planes and subvolumes from the logically contiguous file format.

We measure the read performance using three different types of data organization strategies: Logically Contiguous (LC), linear placement of PGs by the original ADIOS (ORG), and EDO (EDO). Each test case is run 10 times for every data point. The median of top five results is chosen to remove the transient effect.

A. Performance of Planar Reads with PG-level Reordering

We first evaluate the performance of planar reads. In this test case, only PG-level organization using the Hilbert curve is applied by EDO. As shown in Figure 6, the most variation is

observed for linear placement when the stripe count is divisible by 2. Linear placement can achieve better concurrency when the stripe count is a prime number. We set the stripe count to two representative cases, 128 and 137, respectively. Table II shows the theoretical concurrency achievable by different data organizations on three dimensions for two cases: S and X. The numbers are calculated based on the formula we introduced in section IV. The maximum or the exact number of OSTs are listed wherever applicable.

As discussed in section II-A, the read performance is also impacted by the number of seek operations. Table III gives measured numbers of seeks required by each process under different organizations when 64 processes (readers) are used. The numbers may vary when the number of readers changes. ORG and EDO both are based on ADIOS BP format, resulting in identical numbers of seek operations. Because ADIOS uses aforementioned strategy that reads in redundant data to avoid frequent seek operations, it requires much less seeks compared to LC. When 64 (8×8 on one plane) readers read out the data written by 4096 (16×16 on one plane) writers, each reader needs to retrieve 4 (2×2) PGs. Thus 4 seeks are required for each process. We omit one seek operation needed to consult the variable metadata. Because the metadata is only read by the first process, and then passed to the rest of processes. This does not affect the overall analysis.

TABLE II: Concurrency of Planes (number of OSTs)

(a) stripe_count=128

	jk	*ik*	*ij*
EDO (Max/Min)	100/92	104/56	104/74
ORG	128	16	8
LC (S/X)	13/1	10/128	28/128

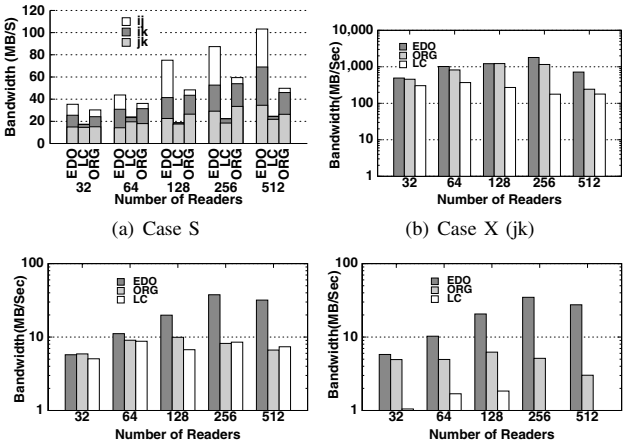
(b) stripe_count=137

	jk	*ik*	*ij*
EDO (Max/Min)	124/115	135/105	134/107
ORG	137	137	137
LC (S/X)	13/1	137/137	137/137

TABLE III: Number of Seeks per Reader (64 Readers)

	S		X	
	LC	ADIOS	LC	ADIOS
jk	40	4	250	4
ik	40	4	250	4
ij	160	4	62500	4

The performance results for a stripe count of 128 are shown in Figure 7. For both cases S and X, LC has better or close performance compared to the other two organizations with small numbers of readers on *jk* planes. This is because LC places these planes as large, sequential data blocks to a few storage targets while the same plane spreads across many OSTs in small units under the other two organizations. Small number of readers also result in more seeks for each process with the ADIOS BP format. Thus, LC particularly favors a small number of reading processes to retrieve data. More processes suffers from contention because read requests



(a) Case S (b) Case X (jk)

(c) Case X (ik) (d) Case X (ij)

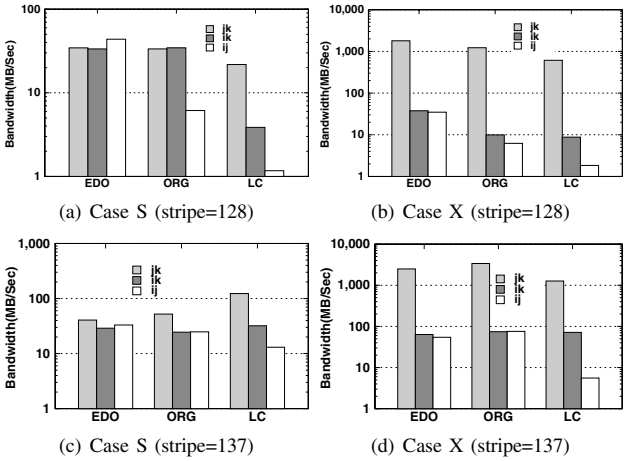
Fig. 7: Planar Performance (writers=4096, stripe=128, plane size (S/X) = 800KB/122MB)

will conflict with each other on a few OSTs. The story is different for *ik* and *ij* planes. Because a large number of seek and read operations are required, and because data spreads across a small number of OSTs, the performance of LC drops significantly. In contrast, ORG and EDO are able to bring the read performance of *ik* and *ij* planes close to that of the *jk* planes in Case S. The decreasing number of OSTs leads to a performance loss to ORG, especially for the *ij* planes. In Case X, ORG performs similarly compared to LC on *ik* and *ij* planes because small concurrency and large amount of read overhead unnecessarily consuming a lot of I/O bandwidth. Even though EDO suffers from the same amount of read overhead on these two dimensions, good concurrency helps it achieve much higher read performance.

Figures 8(a) and 8(b) show the peak performance of two test cases. Good concurrency and fewer seek and read operations help EDO achieve a maximum speedup of 37 times and 7 times for planes of slow-varying dimensions compared to LC and ORG, respectively. We also observe that read overhead of chunking do not impact the performance of EDO for Case S. But distinct performance differences are observed for Case X across three planes.

We conduct the same evaluation when the stripe count is changed to 137 with maximum of 512 readers. We show peak performance in Figures 8(c) and 8(d). Adding more OSTs help three data organizations improve their bandwidth. Particularly, ORG delivers the best performance because of good concurrency as shown in Table II. Even so, EDO is able to achieve performance close to ORG, without sophisticated user efforts in determining that 137 storage targets are optimal for this combination of data size, writing process count and data organization. Same cost of redundant read is observed on Case X, which will be discussed in Section V-B.

In view of its consistent and balanced performance results for all planes with different stripe counts, we believe EDO can be used as a better data organization strategy, particularly beneficial in supporting user convenience and consistent near-



(a) Case S (stripe=128) (b) Case X (stripe=128)

(c) Case S (stripe=137) (d) Case X (stripe=137)

Fig. 8: Performance Differences Across Different Planes (Logscale)

optimal performance.

B. Performance of Intra-PG Ordering for Large Datasets

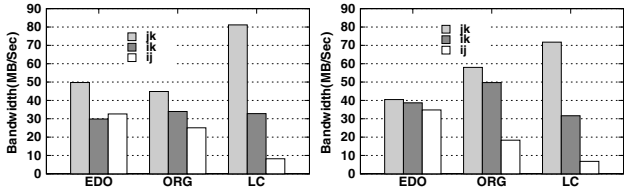
As observed in the previous section, the problem of reading extra data becomes more pronounced when the array size becomes larger. This happens to both ORG and EDO. Thus, on top of PG-level organization using Hilbert Curve, we apply subchunking for intra-PG ordering and then evaluate the performance for planar reads.

Our tests focus on Case X, for which we have observed a great deal of imbalance across different planes. With subchunking, each PG is decomposed into up to 16 subchunks. The main purpose of this approach is to further improve the read performance of ij and ik planes. Thus, only the results for these planes are shown. In Figure 9, subchunking for intra-PG ordering speeds up the read performance further. This is because the finer the PG decomposition is, the less redundant data will be read. Moreover, decomposing PGs reduces the size of each request, which can be served faster by storage devices. However, more seeks are involved with subchunking. Overall, we observe EDO with 8 subchunks delivers the highest bandwidth, achieving an improvement as much as 3 times compared to EDO organization without subchunking (*EDO-1subchunk*), and 22 times compared to ORG. Further decomposition of PG into 16 subchunks degrades the performance due to more seek operations. It also can be attributed to contention at the storage side because of the increasing number of small requests. Last but not least, the intra-PG ordering with 8 subchunks does not increase the time of data generation. For the maximum of 1,024 writers , an increase of 6% to the write time is observed.

C. Scalability of Planar Reads

We also examine the performance of planar reads with an increasing number of writers. In this experiment, the number of readers is fixed to 512, while the data being read is written by different number of processes ranging from 1,024 to 8,192. Thus the amount of data read by each process increases accordingly. Both Hilbert curve and subchunking are used for EDO in this case. The results for Case X are shown in figures 10. Once again EDO is able to maintain good and consistent performance.

D. Planar Reads of Multiple Variables



(a) 36 Readers (b) 72 Readers

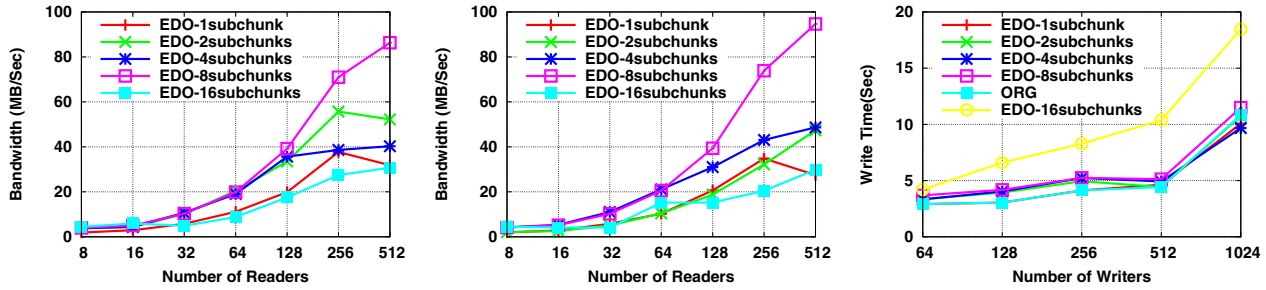
Fig. 11: Read Planes from Multiple Variables

Another common access pattern from scientific codes is to read planes from multiple variables, for example, reading a slice of data from 9 variables out of 14 variables. The ratio of writers to readers is normally $10:1$ or $20:1$. To evaluate the performance of EDO for such patterns, we design a test case that has 8 double precision 3-D arrays written by 720 ($10\times6\times12$) processes. For each variable, a process writes a $22\times36\times22$ 3-D array resulting in a global size of $264\times216\times220$. So the total data size is 765.7 MB. One plane is 445.5 KB. We set the stripe count as 40 and the stripe size as 2 MB so that data created by one process is written to one OST. Only Hilbert curve is used for PG-level organization within EDO.

We evaluate the performance of reading 2-D planes from 5 out of 8 variables. 36 and 72 processes are used, respectively, corresponding to 1/20 and 1/10 of the original 720 writers. The results are shown in Figure 11. LC has the best performance for jk planes. But because of contention, adding more read processes decreases the bandwidth. More seek operations again degrade the performance of ik and ij planes. ORG achieves nearly the same performance on jk and ik planes because their data chunks spread to all 40 OSTs. The difference is due to more extraneous data is retrieved for ik planes. The performance drops significantly for ij planes, especially for 72 processes. This is because such planes are only located on 10 OSTs, compared to the maximum of 35 OSTs using EDO. With 36 processes, the contention is not as severe as that of 72 processes, thus relatively higher bandwidth is observed. EDO performs slightly lower than but comparable to ORG for jk and ik planes. This can be attributed to the difference between the number of OSTs. Overall, EDO delivers consistent and balanced read performance for all planes under both cases and with any choice of striping parameters.

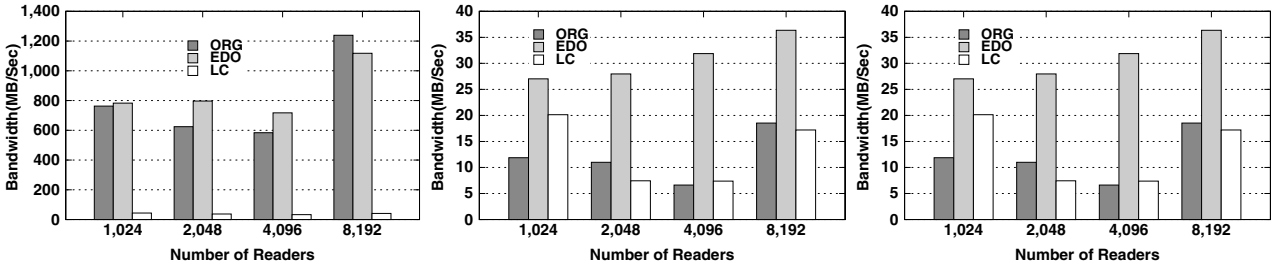
E. Read Performance of Subvolume

A subvolume is an orthogonal, rectangular cube within a multidimensional variable. For these experiments, the small and large data cases are examined using 128 storage targets



(a) ik planes (b) ij planes (c) Data Generation Overhead

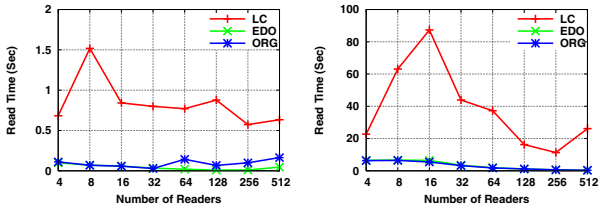
Fig. 9: Benefits of the Hybrid Format of EDO and Data Chunking



(a) jk (b) ik (c) ij

Fig. 10: Planar Performance with Scaling of Writers(Readers=512, stripe=128)

for all three data organizations. To represent an arbitrary subvolume, a volume that containing one-eighth of the total data size is read from the center of the logical simulation area. Each dimension of the subvolume is half of the maximal dimension size. Only Hilbert curve is used for PG-level organization within EDO. Figure 12 shows the experimental results.



(a) Case S (b) Case L

Fig. 12: Subvolume Performance (writers=4096, stripes=128, subvolume size (S/L) = 31.25MB/3.81GB)

In both cases, EDO is able to cover more OSTs than ORG, i.e. 128 and 64, respectively. 64 OSTs are enough to serve up to 512 readers effectively. Thus we observe similar performance between EDO and ORG, with EDO performing slightly better. Even though LC is able to use all OSTs in both cases, its performance suffers because reading a subvolume requires more seek operations. Overall, EDO is able to provide consistent and balanced read performance for planar reads. For cases like reading a subvolume where a dataset covers enough storage targets, EDO is able to achieve the peak read performance.

F. Impact to Data Generation

To examine possible performance impact of EDO to write operations, we evaluate its overhead in terms of time increment to data generation between ORG and EDO. Overall, a maximum of 5% overhead is observed among all the test cases. Detailed results are not included. This test indicates that EDO has negligible impact to the generation of multidimensional datasets in scientific applications.

VI. Related Work

Improving the performance of data-intensive applications has been an active research topic in various domains. Lofstead et al evaluate and understand the performance of many of the reading patterns for extreme scale science applications [26]. Many approaches have explored data staging and caching to either bring data *a priori*, or buffer data temporarily, respectively, in anticipation of performance savings of future data access. For example, staging has been exploited for grid-based scientific computing [22], [7]. The staging approach adopted for grid environments is quite different from what is applicable to closely integrated systems such as supercomputers. The PreDatA [47] system creates a staging area in which data can be prepared through annotation, filtering, indexing, and organization, for efficient post-analysis. Zazen [42] makes intensive use of more storage devices, and caches simulation data as a series of small files across multiple disks of a networked analysis cluster. In doing so, it improves the read performance of data analysis. However, neither PreDatA or Zazen examines the performance of data reordering strategies for I/O performance improvements.

Many studies have investigated different data organizations for boosting I/O performance. For example, log-based data organization is exploited for databases [15] and various file systems [39], [41], [44]. Sarawagi et al. [40] have categorized the strategies for efficient organization of large multidimensional arrays into four classes, namely chunking, reordering, redundancy, and partitioning. Fan et al. [13] proposed a latin cube strategy to put neighbor elements into one shared memory module to improve I/O performance. Because of the natural thinking of the disk traversal, the logically contiguous format has been adopted by many popular I/O libraries including NetCDF versions 3 [32] and 4 [43], HDF5 [28], and PnetCDF [24]. Through the terascale era, this worked extremely well. In fact, HDF5 can achieve excellent performance [46]. With the size and complexity of modern storage arrays, the read performance for 3-D arrays for restart purposes does not measure up for the logically contiguous format, compared to a log-based format [36]. Our work builds on these previous results by examining how SFC-based data organization on storage targets can improve the I/O performance of a common analysis pattern, i.e., reading a plane of different dimensions from multidimensional arrays.

Space filling curves are widely used [3] because of their good spatial locality properties, especially in spatial database researches such as [38], [14]. Lawder et al. [23] have explored different kinds of space filling curves to develop indexing schemes for data layout and fast retrieval in multidimensional databases. Pascucci and Frank [35] have used a global indexing scheme to reorder regular grids based on Lebesgue's space filling curves so that the performance of progressively rendering of multidimensional datasets was improved. Jagadish et al. [19] have examined the linearization of multidimensional data chunks through space filling curves. They have concluded that the Hilbert Curve outperformed all other curves. Moon et al. [30] have demonstrated that the Hilbert Curve can achieve better clustering than both z- and gray-coded curves in two-dimensional and three-dimensional spaces. Hu et al. [18] have used a Hilbert curve for data reorganization and achieved substantial improvements in the performance of fine-grained irregular applications on shared memory systems. Kuo [21] et al. also used Hilbert curve to reorganize the multidimensional datasets. However, they focused on examing the subarray, which is less impacted by the data distribtion, and the scale of study is not comparable to today's petascale systems.

In contrast to previous approaches, the present study uses space filling curves to organize data chunks that naturally arise from the data distribtion in a parallel simulation. As we described in the previous sections, this SFC-based data reordering approach achieves many performance benefits when reading complex data elements from a multidimensional datasets without causing performance losses when writing the same datasets.

VII. Conclusions

As an extension of ADIOS, we have designed and developed EDO (Elastic Data Organization) for efficient data retrieval from scientific multidimensional datasets. Multiple organization strategies are supported as selectable algorithms for data ordering at two different levels. The first level uses Hilbert curve for distributing and ordering ADIOS data groups. By using this strategy, data from scientific multidimensional arrays can be distributed in a balanced manner across all storage devices, so that the aggregated bandwidth can be effectively aggregated and exploited for challenging read access patterns, particularly planar reads. A subchunking strategy is also introduced for data ordering at the intra-PG level, splitting a large data chunk into many small subchunks. This helps achieve a good tradeoff between the number of seeks and the amount of read overhead for large data datasets. Together, EDO is able to deliver consistent and balanced read performance for all types of planar reads from multidimensional arrays. We have mathematically validate the performance benefits of EDO on large-scale parallel file systems. We have also experimentally evaluated the performance of EDO using the Jaguar supercomputer at Oak Ridge National Laboratory. Our results demonstrate that EDO can improve the performance of planar reads by as much as 37 times.

In the future, we plan to exploit more data ordering techniques to integrate into EDO for other access pattern such as multi-resolution arrays. We also plan to study the data organization strategies on the other file systems that do not support changing the stripe parameters on the fly such as GPFS.

VIII. Acknowledgments

This work is funded in part by a UT-Battelle grant to Auburn University, and in part by National Science Foundation awards CNS-1059376 and CNS-0917137. This research is sponsored in part by the Office of Advanced Scientific Computing Research; U.S. Department of Energy; and Sandia National Laboratories under contract DE-AC04-94AL85000. This research used resources of the Argonne Leadership Computing Facility at Argonne National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-06CH11357.

References

[1] H. Abbasi, G. Eisenhauer, M. Wolf, and K. Schwan. Datastager: scalable data staging services for petascale applications. In *HPDC '09*, New York, NY, USA, 2009. ACM.

[2] H. Abbasi, J. Lofstead, F. Zheng, K. Schwan, M. Wolf, and S. Klasky. Extending i/o through high performance data services. In *Cluster Computing and Workshops, 2009. CLUSTER '09. IEEE International Conference on*, pages 1 –10, 31 2009-sept. 4 2009.

[3] T. Asano, D. Ranjan, T. Roos, and E. Welzl. Space filling curves and their use in the design of geometric data structures. In *Lect. Notes Comput. Sc.*, volume 911, pages 6–44. 1995.

[4] L. Chacón. A non-staggered, conservative, $\nabla \dot{s} B \rightarrow = 0$, finite-volume scheme for 3D implicit extended magnetohydrodynamics in curvilinear geometries. *Computer Physics Communications*, 163:143–171, Nov. 2004.

[5] C. S. Chang, S. Klasky, J. Cummings, R. Samtaney, A. Shoshani, L. Sugiyama, D. Keyes, S. Ku, G. Park, S. Parker, N. Podhorszki, H. Strauss, H. Abbasi, M. Adams, R. Barreto, G. Bateman, K. Bennett, Y. Chen, E. D. Azevedo, C. Docan, S. Ethier, E. Feibush, L. Greengard, T. Hahm, F. Hinton, C. Jin, A. Khan, A. Kritz, P. Krsti, T. Lao, W. Lee, Z. Lin, J. Lofstead, P. Mouallem, M. Nagappan, A. Pankin, M. Parashar,

M. Pindzola, C. Reinhold, D. Schultz, K. Schwan, D. Silver, A. Sim, D. Stotler, M. Vouk, M. Wolf, H. Weitzner, P. Worley, Y. Xiao, E. Yoon, and D. Zorin. Toward a first-principles integrated simulation of tokamak edge plasmas - art. no. 012042. *Scidac 2008: Scientific Discovery through Advanced Computing*, 125:12042–12042, 2008.

[6] J. H. Chen et al. Terascale direct numerical simulations of turbulent combustion using S3D. *Comp. Sci. & Disc.*, 2(1):015001 (31pp), 2009.

[7] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, and S. Tuecke. The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets. *J. Netw. Comput. Appl.*, 23:187–200, 1999.

[8] H. Childs. Architectural challenges and solutions for petascale postprocessing. *J. Phys.*, 78(012012), 2007.

[9] Cluster File System, Inc. Lustre: A Scalable, High Performance File System. http://www.lustre.org/docs.html.

[10] Y. Cui, K. B. Olsen, T. H. Jordan, K. Lee, J. Zhou, P. Small, D. Roten, G. Ely, D. K. Panda, A. Chourasia, J. Levesque, S. M. Day, and P. Maechling. Scalable earthquake simulation on petascale supercomputers. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10, pages 1–20, Washington, DC, USA, 2010. IEEE Computer Society.

[11] C. H. Q. Ding and Y. He. Data organization and I/O in a parallel ocean circulation model. In *Proc. SC99*. Society Press, 1999.

[12] C. Docan, F. Zhang, M. Parashar, J. Cummings, N. Podhorszki, and S. Klasky. Experiments with memory-to-memory coupling for end-to-end fusion simulation workflows. *Cluster Computing and the Grid, IEEE International Symposium on*, 0:293–301, 2010.

[13] C. Fan, A. Gupta, and J. Liu. Latin cubes and parallel array access. In *Parallel Processing Symposium, 1994. Proceedings., Eighth International*, pages 128 –132, apr 1994.

[14] G. Ghinita, P. Kalnis, and S. Skiadopoulos. Prive: anonymous location-based queries in distributed mobile systems. In *Proceedings of the 16th international conference on World Wide Web*, WWW '07, pages 371–380, New York, NY, USA, 2007. ACM.

[15] J. Gray et al. The recovery manager of the system R database manager. *ACM Comput. Surv.*, 13(2):223–242, 1981.

[16] R. W. Grout, A. Gruber, C. Yoo, and J. Chen. Direct numerical simulation of flame stabilization downstream of a transverse fuel jet in cross-flow. *P. Combust. Inst.*, 2010. In press.

[17] D. Hilbert. Ueber die stetige abbildung einer line auf ein flächenstück. *Math. Ann.*, 38:459–460, 1891.

[18] Y. Hu, A. Cox, and W. Zwaenepoel. Improving fine-grained irregular shared-memory benchmarks by data reordering. In *Proc. SC00*, pages 33 – 33, Nov. 2000.

[19] H. V. Jagadish. Linear clustering of objects with multiple attributes. *SIGMOD Rec.*, 19(2):332–342, 1990.

[20] S. Klasky, S. Ethier, Z. Lin, K. Martins, D. McCune, and R. Samtaney. Grid -based parallel data streaming implemented for the gyrokinetic toroidal code. In *SC '03: Proceedings of the 2003 ACM/IEEE conference on Supercomputing*, page 24, Washington, DC, USA, 2003. IEEE Computer Society.

[21] S. Kuo, M. Winslett, Y. Cho, J. Lee, and Y. Chen. Efficient input and output for scientific simulations. In *In Proceedings of I/O in Parallel and Distributed Systems (IOPADS)*, pages 33–44. ACM Press, 1999.

[22] E. Laure, H. Stockinger, and K. Stockinger. Performance engineering in data grids. *Concurr. Comp-Pract. E.*, 17:4–171, 2005.

[23] J. K. Lawder and P. J. H. King. Querying multi-dimensional data indexed using the hilbert space-filling curve. *SIGMOD Rec.*, 30(1):19–24, 2001.

[24] J. Li et al. Parallel netCDF: A high-performance scientific I/O interface. In *Proc. SC03*. ACM, 2003.

[25] J. Lofstead et al. Managing variability in the io performance of petascale storage system. In *Proc. SC10*, New York, NY, USA, 2010. IEEE Press.

[26] J. Lofstead, M. Polte, G. Gibson, S. Klasky, K. Schwan, R. Oldfield, and M. Wolf. Six degrees of scientific data: Reading patterns for extreme scale science io. In *Proceedings of the 20th International ACM Symposium on High-Performance Parallel and Distributed Computing, to appear*, HPDC'11. ACM, 2011.

[27] J. Lofstead, F. Zheng, S. Klasky, and K. Schwan. Adaptable, metadata rich IO methods for portable high performance IO. *Parallel and Distributed Processing Symposium, International*, 0:1–10, 2009.

[28] The HDF Group. Hierarchical data format version 5, 2000–2010. http://www.hdfgroup.org/HDF5.

[29] O. E. B. Messer, S. W. Bruenn, J. M. Blondin, W. R. Hix, A. Mezzacappa, and C. J. Dirk. Petascale supernova simulation with chimera. *Journal of Physics: Conference Series*, 78, July, 2007.

[30] B. Moon, H. Jagadish, C. Faloutsos, and J. Saltz. Analysis of the clustering properties of the hilbert space-filling curve. *IEEE T. Knowl. Data En.*, 13(1):124–141, 2001.

[31] NCCS. http://www.nccs.gov/computing-resources/jaguar/.

[32] NetCDF. http://www.unidata.ucar.edu/software/netcdf/.

[33] R. Niedermeier and P. Sanders. On the manhattan-distance between points on space-filling mesh-indexings. Technical report, 1996.

[34] R. Oldfield, P. Widener, A. Maccabe, L. Ward, and T. Kordenbrock. Efficient data-movement for lightweight i/o. *Cluster Computing, IEEE International Conference on*, 0:1–9, 2006.

[35] V. Pascucci and R. J. Frank. Global static indexing for real-time exploration of very large regular grids. In *Proc. SC01*, pages 45–45, Nov. 2001.

[36] M. Polte et al. ...and eat it too: High read performance in write-optimized HPC I/O middleware file formats. In *In Proceedings of Petascale Data Storage Workshop 2009 at Supercomputing 2009*, 2009.

[37] F. Pretorius. Evolution of Binary Black Hole Spacetimes. *Phys. Rev. Lett.*, 95:121101, 2005.

[38] J. A. Rice. *Hilbert R-tree: An improved R-tree using fractals*. Duxbury Press, Ibrahim Kamel, Christos Faloutsos, 1995.

[39] M. Rosenblum and J. K. Ousterhout. The design and implementation of a log-Structured file system. *ACM T. Comput. Syst.*, 10:1–15, 1991.

[40] S. Sarawagi and M. Stonebraker. Efficient organization of large multidimensional arrays. In *Proc. 10th Int. Conf. on Data Eng.*, pages 328–336, Houston, TX, 1994.

[41] M. Seltzer, K. Bostic, M. K. Mckusick, and C. Staelin. An implementation of a log-Structured file system for UNIX. In *USENIX'93*, pages 3–3, Berkeley, CA, USA, 1993. USENIX Association.

[42] T. Tu et al. Accelerating parallel analysis of scientific simulation data via zazen. In *FAST'10*, pages 129–142. USENIX Association, 2010.

[43] Unidata. http://www.hdfgroup.org/projects/netcdf-4/.

[44] R. Y. Wang, T. E. Anderson, and D. A. Patterson. Virtual log based file systems for a programmable disk. In *OSDI '99: Proceedings of the third symposium on Operating systems design and implementation*, pages 29–43, Berkeley, CA, USA, 1999. USENIX Association.

[45] W. X. Wang and et al. Gyro-kinetic simulation of global turbulent transport properties in Tokamak experiments. *Physics of Plasmas*, 13(9):092505, 2006.

[46] W. Yu, J. Vetter, and H. Oral. Performance characterization and optimization of parallel I/O on the cray XT. *IPDPS*, pages 1–11, April 2008.

[47] F. Zheng et al. Predata - preparatory data analytics on peta-scale machines. In *IPDPS*, Atlanta, GA, April 2010.