2012 IEEE 20th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems

SMART-IO: SysteM-AwaRe Two-Level Data Organization for Efficient Scientific Analytics

Yuan Tian[1] Scott Klasky[2] Weikuan Yu[1] Hasan Abbasi[2] Bin Wang[1]
Norbert Podhorszki[2] Ray Grout[3] Matthew Wolf[4]

Auburn University[1]
{*tianyua,wkyu,bzw0012*}*@auburn.edu*

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

National Renewable Energy Laboratory[3]
Ray.Grout@nrel.gov

Georgia Institute of Technology[4]
mwolf@cc.gatech.edu

***Abstract*—Current I/O techniques have pushed the write performance close to the system peak, but they usually overlook the read side of problem. With the mounting needs of scientific discovery, it is important to provide good read performance for many common access patterns. Such demand requires an organization scheme that can effectively utilize the underlying storage system. However, the mismatch between conventional data layout on disk and common scientific access patterns leads to significant performance degradation when a subset of data is accessed. To this end, we design a system-aware Optimized Chunking model, which aims to find an *optimized* organization that can strike for a good balance between data transfer efficiency and processing overhead. To enable such model for scientific applications, we propose SMART-IO, a two-level data organization framework that can organize the blocks of multidimensional data efficiently. This scheme can adapt data layouts based on data characteristics and underlying storage systems, and enable efficient scientific analytics. Our experimental results demonstrate that SMART-IO can significantly improve the read performance for challenging access patterns, and speed up data analytics. For a mission critical combustion simulation code S3D, Smart-IO achieves up to 72 times speedup for planar reads of a 3-D variable compared to the logically contiguous data layout.**

***Keywords*-Smart-IO; ADIOS; Data Organization; Parallel I/O; S3D**

I. Introduction

The increasing growth of leadership computing capabilities, both in terms of complexity as well as computational power, has enabled scientific applications to solve complex scientific problems at large scale. This is accompanied by a gigantic volume of complex data produced, driving data intensive computing as a propelling task force for scientific discovery. The size of dataset is typically on the order of terabytes, growing towards petabytes. As a major portion of application turnaround time, I/O plays a significant role in determining simulation productivity and energy efficiency.

Many efforts, both past and present, have heavily focused on improving the I/O performance by studying the write side of the problem, but the read performance of scientific applications on large-scale systems has not received the same level of attention, despite its importance to drive scientific insight through scientific simulation, analysis workflows and visualization. Worse yet, current I/O techniques often overlook the need of good read performance. The major reason for this discrepancy between writes and reads is due to the physical limitations of magnetic storage and the common access patterns of scientific applications. Physical disks are optimized for one-dimensional large sequential blocks of data while scientific datasets are usually multidimensional. Data elements from a multidimensional scientific dataset are usually stored to a one dimensional physical disk space based on the order of one primary dimension. This results in noncontiguous placement of data elements on secondary and tertiary dimensions. When reading the data elements in the order of higher dimensions, the performance degrades significantly due to the expense of coping with noncontiguity, either because of extra disk seeks or because of extra retrieved data. In addition, the peak aggregated bandwidth of parallel storage systems cannot be effectively utilized when only a subset of the data is requested. This occurs because the requested data is concentrated on a very small number of storage targets with current data placement strategies, causing a substantial performance degradation and limited scalability. Systems such as the Gordon [2] supercomputer at San Diego Supercomputer Center attempt to address this problem through new hardware such as solid-state devices (SSDs). However, without optimizing data organizations, even SSDs cannot achieve an efficient utilization of its peak disk bandwidth.

To enable efficient I/O, another issue that needs to be addressed is the complexity of the simulation output. One simulation dataset is normally a collection of many multidimensional variables. Each variable possesses distinct characteristics. For example, a simulation may generate one variable that is on the order of hundreds of gigabytes, while another variable is only hundreds of megabytes. After domain decomposition, a common parallelization strategy in scientific applications, such differences become even more pronounced. It is difficult to achieve the optimal performance by applying a uniform data organization strategy to all variables. Therefore, a technique that can dynamically organize each variable to match with the underlying storage system is desired.

To address the aforementioned issues, we have introduced

1526-7539/12 $26.00 © 2012 IEEE
DOI 10.1109/MASCOTS.2012.30

a theoretical **Optimized Chunking** model to derive the *optimized* chunk size and guide data organization during the simulation run-time. Such a model is established upon a collection of system parameters. So the organization is able to adapt to the underlying system. To enable optimized chunking, we propose a two-level data organization scheme. The first level is focused on the construction of *optimized* sized chunks. A Space Filling Curve based data placement strategy is used to ensure near-maximum data concurrency at the second level. We have implemented this multi-level strategy as a light-weight middleware called **Smart-IO**. Smart-IO has been tested and evaluated as a part of the ADaptable I/O System (ADIOS) framework [1]. We use ADIOS to leverage its penetration amongst existing scientific applications. For our experimental evaluation we use the Jaguar Cray XT5 [3] supercomputer at Oak Ridge National Laboratory(ORNL) to demonstrate that Smart-IO is able to achieve both good balance and high performance for reading some of the challenging access patterns of scientific applications. In our test case for S3D, a mission-critical combustion code, we obtain a 72x speedup to the planar read performance compared to the logically contiguous data layout, while introducing negligible overhead in data generation.

The rest of the paper is organized as follows. We first discuss the background and motivation of this work in detail in Section II. We then introduce the Optimized Chunking model in Section III. The design details of Smart-IO are described in Section IV. Section V validates our strategy through a comprehensive set of experimental results. A survey of literature review is provided in Section VI. Finally, we conclude the paper in Section VII.

II. Background and Motivation

In order to improve the read performance for scientific applications, a thorough understanding of application access patterns is crucial. This is because the physical limitation of magnetic disks where varying access patterns can result in substantial changes in the performance of reads. Two scenarios are highly important when optimizing read performance for end-to-end simulation workflows: 1) checkpoint-restart, for which a fraction of the memory image is read from storage in order to recover from failure; 2) data post-processing, for which a smaller data subset is read in for the purpose of visualization or data analytics. In our previous work [30], we have identified major access patterns into three categories.

- Read in **all** of a single variable
- Read an arbitrary orthogonal **subvolume**
- Read an arbitrary orthogonal **full plane**

More complex reading patterns can be described through a composition of these three patterns, or through minor variations. Figure 1 gives an example of a 3-D array, a 3-D subvolume in the center of the array, and three 2-D planes in three dimensions: i, j, and k; where i is the primary, i.e., the slowest varying dimension, and k is the tertiary i.e., the fastest varying dimension. Data in the array is stored first along the fastest dimension k, then along the slower dimensions, j and i, on disk.

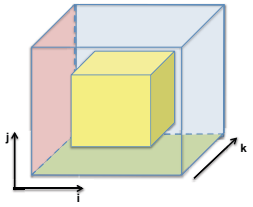


Fig. 1: Data Organization and Performance

Among these patterns, significant performance variations are observed for range queries on the data subsets, particularly on the orthogonal planes. Such phenomenon is often referred as "dimension dependency" [28], where the read performance depends on the dimensions of the query, rather than the data size of the query. Despite the performance concerns, this access pattern is commonly used by scientific post-processing [9].

A. Optimizing the Chunk Size

Our previous work [30], [29] have shown that utilizing a Space Filling Curve-based (SFC-based) data placement strategy can obtain a high degree of concurrency of storage access for common access patterns of scientific analytics. Such strategy is based on the *chunking* data organization, which outperforms logically contiguous (LC) as it is able to alleviate the dimension dependency [28]. However, the impact of the chunk size on read performance can create a severe bottleneck. For example, if an application with a uniform domain decomposition outputs multiple variables with distinct sizes, a large number of different segments in the logical file will be produced. For the smaller sized variables, the performance of reading is bottleneck by the excessive seeks required to retrieve the large number of small data chunks. For larger variables, the dimension dependency remains a significant bottleneck when reading data on the *slow* dimensions, as the layout forces the I/O subsystem to read a significant amount of unnecessary segment gaps in order to complete the request without excessive seeking. Thus it is an important concern to find the correct chunk size for the system.

To address the aforementioned challenges, we conducted an initial study in [31]. In this paper we propose a simplified model to describe the effect of the chunk size on the read access time. We have also introduced a two-level data organization scheme that combines this chunk size optimization with the SFC-based data layout optimization to significantly improve read performance for typical access patterns. Next we describe the development of this model as well as the selection of an *optimized* chunk size.

III. Optimized Chunking

In order to find the *good* data organization, we first need to theoretically investigate what is the *correct* chunk size for multidimensional data on a parallel storage system. The I/O performance on a large-scale system is influenced by many

factors, such as the communication cost, the number of writers, the size of the chunks, the number of storage targets, etc. As mentioned earlier, for a chunk based data organization, the size of chunks plays a critical role in determining the read performance for query on a data subset where data points are not contiguously stored. Thus optimizing read performance requires a model to find an effective chunk size that works well under the constraints of HPC system characteristics.

In general, the response time for a read request on a large scale system can be expressed in terms of five parameters,

$$RT = (T_{comm} + T_{seek} + T_{io} + T_{local}) \times \alpha$$

Here T_{comm} is the time for each client to send out the request to the storage system over the network. T_{seek} is the total time to perform seek operations to locate the data on disk. It is the sum of the seek time, disk head settling time and rotational latency, which are all dependent on the physical characteristics of the disk and the physical location of the data on the disk. Without taking into account the physical placement of data on a particular disk, we use the return time of seek calls at the file system level to quantify the seek time. T_{io} is the time taken to read out all the requested data from disk. Since storage systems are commonly attached to the computation partition as separate units, this time is taken from the perspective of the client. Thus we include the data transfer time for the network to return data. The last parameter, T_{local}, is the time that a client takes to perform local processing such as the memory copy. T_{local} depends on the contiguity of requested data in memory, but is significantly smaller than the other costs of I/O. α represents the interference factor on the large-scale system. However, to determine this factor needs a thorough study of the system and it is not the focus of this work. For a simplified analytical model, the external and internal interferences to the storage system are ignored. Such finely guided modeling can help pinpoint a solution that enables near-optimal I/O performance tuning in a timely fashion.

Given the read time as describe by Equation (1), we now expand the model to calculate a chunk size that can provide near-optimal performance. Assume this optimized chunk size is **OCS** and the original chunk size is **CS**. A data chunk that is larger than OCS needs to be divided, while smaller data chunks require aggregation to the proper size. Therefore, we have

$$OCS = \begin{cases} \frac{CS}{N_{ocs}}, \text{when dividing large chunks} \\ CS \times N_{ocs}, \text{when aggregating small chunks,} \end{cases} \quad (1)$$

where N_{ocs} is the number of chunks of size **OCS**.

Assume a read request contains K optimized chunks, according to Equation (1), the read time RT on the slow dimension(s) can be expressed as

$$RT_{slow} = K \times (CC + T_s + \frac{OCS}{BW_{io}} + T_{local}), \quad (2)$$

where CC is the communication cost unit, $BW_{i/o}$ is the I/O bandwidth, T_s is the time unit for each seek operation, and T_{local} is time to perform memory copy operation for each chunk. The relationship between OCS, K, N_{ocs} and T_{seek} can be described as $OCS \propto \frac{1}{K} \propto \frac{1}{N_{ocs}} \propto \frac{1}{T_{seek}}$. Since we do not have a numerical relationship between K and N_{ocs}, there are two scenarios to be considered separately in order to solve Equation (2).

I/O Bounded: When OCS is too large, the time spent on I/O outweighs the cost of communication and seek operations. In this case, the entire read time is dominated by T_{io}, resulting in T_{seek} and T_{cc} being obviated in the expression. The read time on the slow dimension RT_{slow} can be represented as:

$$RT_{slow} = K \times (\frac{OCS}{BW_{io}} + T_{local}), \quad \text{when } K \times (CC + T_s) << \frac{K \times OCS}{BW_{io}}. \quad (3)$$

Seek Bounded: When OCS is too small, i.e., N_{ocs} is large, because $N_{ocs} \propto T_{seek}$, response time is more dominant by T_{seek}, and T_{io} will not appear in the expression. Therefore, we have

$$RT_{slow} = K \times (CC + T_s + T_{local}), \quad \text{when } K \times (CC + T_s) >> \frac{K \times OCS}{BW_{io}}. \quad (4)$$

Notice the combination of Equations (3) and (4) can be visualized using the dark red curved line in Figure 2. The left part of the curve represents the Equation (3), while the right side of the curve represents the Equation (4).

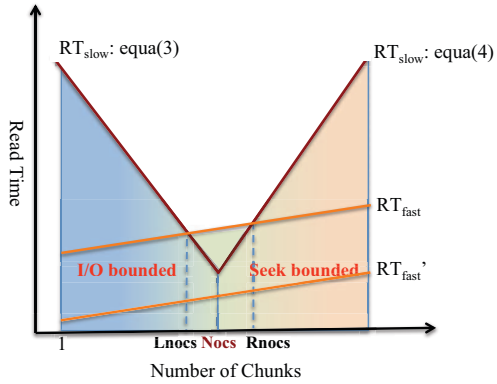


Fig. 2: The Read Time vs. the Number of Chunks

The function's minimum is achieved at the point of discontinuity, that is, when:

$$K \times (CC + T_s) = \frac{K \times OCS}{BW_{io}} \quad (5)$$

Canceling K out from both sides of the equation, we have the Optimized Chunks Size:

$$OCS = BW_{io} \times (CC + T_s) \quad (6)$$

Correspondingly, N_{ocs} can be calculated as:

$$N_{ocs} = \begin{cases} \frac{CS}{BW_{io} \times (CC + T_s)}, \\ \text{when dividing large data chunks} \\ \\ \frac{BW_{io} \times (CC + T_s)}{CS}, \\ \text{when aggregating small data chunks} \end{cases} \quad (7)$$

It should be noted that the strategy of chunking is to improve the read performance when accessing the slow dimension(s) by sacrificing the performance on the fast dimension. This may cause the read performance on the fast dimension to become slower than the read performance on the slow dimension! This relationship is represented in Figure 2 by two orange straight lines. As mentioned earlier, in view of the general performance for any access pattern, the fastest total response time may not occur at the point N_{ocs} but still within the *Optimized Region* of L'_{nocs} and R_{nocs}". The performance difference inside the Optimized Region is within a small range. As this study is aimed at finding an optimized chunk size, we use our solution of N_{ocs} as the guidance for data organization. As presented later, our experimental results demonstrate that this value provides satisfactory performance.

IV. Smart-IO: Two-Level System-Aware Data Organization

To enable the Optimized Chunking model for efficient scientific data analytics, we introduce Smart-IO, a light-weighted software framework that can dynamically construct the multidimensional scientific data into an optimized organization. Figure 3 shows the software architecture of Smart-IO and its components.

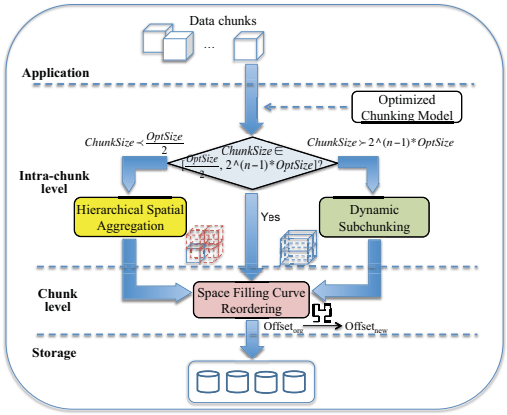


Fig. 3: Two-Level Data Organization of Smart-IO

From high level, Smart-IO sits between the application layer and the storage system. It provides a two-level data organization. The first level is intra-chunk level, which focuses on building the data chunks into *optimized* size (OptSize). OptSize is a system-aware value derived from our *Optimized Chunking* model, which can achieve a good balance between the data transfer efficiency and processing overhead through specific system parameters. For a data chunk that does not satisfy the OptSize, it needs to be reconstructed accordingly. To serve this purpose, two strategies are designed, namely *Hierarchical Spatial Aggregation* (HSA) and *Dynamic Subchunking* (DYS). At the second level, which is the chunk level, a default SFC (Space Filling Curve)-based reordering is used to distribute data chunks among storage devices to enable good data parallelism. Under such organization, a data chunk has three paths moving towards the storage system, as shown in Figure 3.

As we can see, a decision window is constructed as $[\frac{OptSize}{2}, OptSize \times 2^{n-1}]$. The rational of this window is provided in section IV-D to determine the path of the data chunk. Essentially the goal of the HSA and DYS is to construct data chunks that fall into the window, where the SFC-based chunk ordering is applied. The rest of this section describes the design of these components in detail.

A. Hierarchical Spatial Aggregation

Even though the scientific applications normally generate a gigantic amount of data, it is not rare for an output dataset contains one or few small variables. A small variable is turned into even smaller pieces after domain decomposition. A significant number of seeks and memory operations are required for common access patterns, correspondingly leading to limited read performance. Aggregation is a technique that widely used to converge small pieces of data. However, simply concatenating small chunks does not solve the problem. Because the number of disk and memory operations remains the same for reading. Thus, we design Hierarchical Spatial Aggregation which aggregates data chunks in a way that their spatial localities are reserved. For every spatially adjacent 2^n processes, an *Aggregation Group* (AG) is formed. Within each AG, one process is selected as the aggregator for one variable. If there is more than one variable needs to be aggregated, the aggregator process will be selected in a round-robin fashion within the same group to achieve load balancing. The aggregator aggregates data from 3 closest neighboring processes and constructs them into a larger data chunk. If the aggregated chunk size still does not fall into the decision window for output, a second level of aggregation will be performed among the first level aggregators who have hold all the data in their memories. Figure 4 shows an example of aggregating one variable from 16 processes in a 2-D space. For every spatially adjacent 4 processes, an *Aggregation Group* is formed. Process 0 is selected as the first aggregator. In our case, process 0 is chosen as the second level aggregator. After aggregation, only the aggregators will be writing out the data. Figure 5 gives an example of data movement and file output for 3 variables where 2 of them, var2 and var3, qualify for the HSA. After HSA, only process 0 needs to write out var2 and process 1 needs to write out var3. With HSA, the amount of requests and seek operations are reduced by $level \times 2^n$ times, where level is the level of HSA performed.

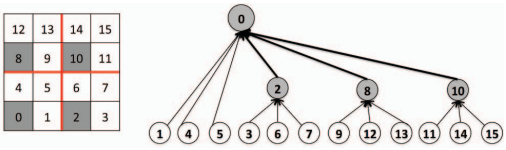


Fig. 4: Hierarchical Spatial Aggregation

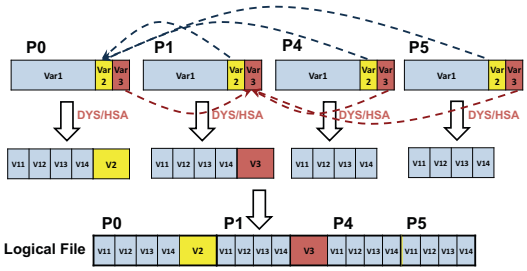
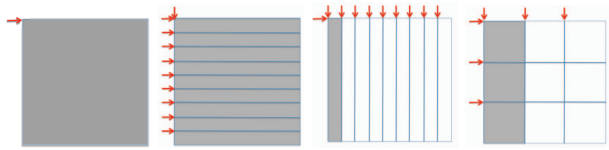


Fig. 5: Data Movement and File Output

B. Dynamic Subchunking

Chunking has shown its capability to alleviate the *dimension dependency* for reading. However, because data points are laid out sequentially within a chunk, dimension dependency can be significant again when a data chunk itself is large. A range query on a slow dimension either has to perform a large amount of seek operations, or to read in a lot of redundant data from the start to the end point of request, as shown in Figure 6(a). Even though the latter option is more preferable on parallel file system, fundamentally neither approach is efficient. As OptSize provides the good balance between the size of data transfer and the processing overhead, each large data chunk can be decomposed into *subchunks* with the size of OptSize to further alleviate dimension dependency. Subchunking the large data chunk is important for applications that access data with a high degree of locality.



(a) Original (b) option 1 (c) DYS (d) option 3

Fig. 6: Design of Dynamic Subchunking

However, how to decompose a chunk needs to be investigated. Figure 6(b) to Figure 6(d) provide examples of three common options for subchunking. The red arrow represents seek operation. The shaded region represents the amount of data needs to be read in for a request on slow dimension. The row major is the fast dimension and column major is the slow dimension. As we can see, subchunking on the slow dimension does not benefit reading on that dimension instead of introducing more seek operations. The amount of data overhead is determined by the amount of subchunking on the fast dimension, which also is proportional to the performance degradation on the fast dimension. However, reading on fast dimension normally is always efficient compared to the slow dimension(s) because data is laid out contiguously. For example, reading 120MB data on Jaguar is expected to cost less than half a second with one process. Assume the read time is proportional to the number of seeks, which normally can be optimized by using more readers, subchunking into 9 subchunks along the fast dimension will increase the read time to 4.5 seconds. This is still within the tolerable margin, compared to more than 60 seconds for read time on the slow dimension as measured. Thus, we choose option 2 as subchunking distribution strategy for our design. Essentially, *subchunking will be performed on all the dimensions except the slowest dimension for an n-dimensional data chunk. The number of subchunks on each dimension will be well balanced.*

In the following sections, we use a series of numbers X_1-X_2...-$X_{(n)}$ to represent the number of subchunks on an n-dimensional array, where n represents the slowest dimension.

C. Data Organization based on Space Filling Curve

After the Optimized Chunks are constructed, a Hilbert Space Filling Curve ordering is used to rearrange the placement of data chunks on storage. The rational of such strategy is based on our earlier work [30].

D. Optimized Chunk Size Decision Window

For a n-dimensional array, the decision window is constructed as $[\frac{OptSize}{2}, OptSize \times 2^{n-1}]$. The upper bound of this window is decided by our subchunking strategy. Because we only partition data on n-1 dimensions as described in IV-B, and the minimum partition on each dimension is 2. Therefore given a data chunk, the minimum number of subchunks is 2^{n-1}, leading to the minimum chunk size qualifies for subchunking to be $OptSize \times 2^{n-1}$. On the other hand, the Hierarchical Spatial Aggregation is performed among the closest neighbors on n dimensions, leading to minimum 2^n chunks to be aggregated. However, we do not want to over aggregate data chunks that result in a chunk size where subchunking is required. As the upper bound of the chunk size is $OptSize \times 2^{n-1}$. So we have the lower bound as $\frac{OptSize \times 2^{n-1}}{2^n}$, which is $\frac{OptSize}{2}$.

V. Experimental Results

We have implemented Smart-IO within ADIOS, an I/O middleware from ORNL that has been used by a number of scientific applications [4], [14], [16], [36], [20] for optimized I/O performance. By default ADIOS applies chunking for multidimensional arrays. We evaluate Smart-IO on the *Jaguar* supercomputer at ORNL. Jaguar is currently the third fastest supercomputer in the world [17]. It is equipped with Spider (an installation of Lustre) for the storage subsystem. In our experiments, we used Widow 2 partition of Spider which contains 336 storage targets (OSTs).

The I/O bandwidth on Spider is approximately 250MB/Sec per OST, the average seek time is 8ms, and the communication cost is about 1.9ms. Thus, the OCS is calculated

as 2.5MB using Equation (6). The decision window size is $[1.25MB, 10MB]$ for Optimized Chunking policy. Based on the previous practices of the ADIOS team on Jaguar, the stripe size is set as the size of ADIOS process group. This practice can maximize data concurrency, reduce false sharing on the Lustre file system, and reduce the internal and external interferences [15].

S3D [7] combustion simulation code from Sandia National Laboratories is used in our experiments. It is a high-fidelity, massively parallel solver for turbulent reacting flows. S3D employs a 3-D domain decomposition to parallelize the simulation space. We set up the output file to contain 4 variables (Var1, Var2, Var3 and Var4) with different sizes. Table I shows the data chunk size after the original 3-D domain decomposition and the exemplary variable sizes with 4,096 ($16 \times 16 \times 16$) processes. The Smart-IO operations performed on each chunk based on the decision window are also listed.

TABLE I: Test Variables (Elements/Size)

	Var1	Var2	Var3	Var4
Chunk	256^3/128MB	128^3/16MB	64^3/2MB	32^3/256KB
Variable	4096^3/512GB	2048^3/64GB	1024^3/8GB	512^3/1GB
Operations	DYS/SFC	DYS/SFC	SFC	HSA/SFC

The performance evaluation of Smart-IO is mainly focused on the I/O performance of planar read, which is the most common yet very challenging access pattern. We measure the read performance among three types of data organization strategies: Logically Contiguous (LC), the chunking strategy of the original ADIOS (ORG), and two-level data organization of Smart-IO (Smart). A separate test program is created to evaluate the I/O performance of logically contiguous data layout. Each test case is run 10 times for every data point. The median of top five numbers is chosen as the result.

A. Data Generation

One of the design considerations of Smart-IO is to constrain the performance impact on data generation. As shown in Table I, Dynamic Subchunking is performed on Var1 and Var2, while one level of Hierarchical Spatial Aggregation is performed on Var4. We evaluate the write time of using 4,096 processes to output the entire file. The total output time along with the time breakdown is shown in Table II.

TABLE II: Write Time Break Down

	I/O	Subchunking	Aggregation	Total
ORG	41.49	0	0	41.49
Smart	41.67	0.87	0.13	42.67

As we can see, subchunking and aggregation do not cause significant delays to the write time. Only 2.8% overhead is observed to the total write time. We also evaluate the weak scaling of data generation. As shown in Figure 7, very limited overhead is introduced in all cases.

B. Planar Read with Dynamic Subchunking

As the decision of Dynamic Subchunking is based on the value of OCS. Thus, we evaluate the performance of

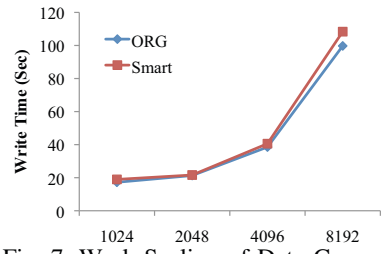


Fig. 7: Weak Scaling of Data Generation

subchunking to assess the accuracy of our algorithm. Based on the OCS value and Equation (6), subchunking is performed on Var1 and Var2. As both cases exhibit the same behavior, we only show the test results for Var1 as a representative case. With Smart-IO, a 7-7-1 (49 total) subchunking is performed on Var1. To evaluate the accuracy of our model, we then vary the number of subchunks on each dimension within a range of $[N_{OCS}-2, N_{OCS}+2]$, that is $[5, 9]$ in our case. This leads to four other different number of subchunks 25, 36, 64 and 81. A planar read is performed on each of three dimensions. The number of readers varies from 32 to 512, as to follow the tradition that application scientists often spend only 10% of the writers to read. The total read time on three dimensions are shown in Figure 8. The number of X-axis represents the number of chunks on k, j, and i dimensions respectively, while O-7-7-1 represents the value calculated from the OCS model.

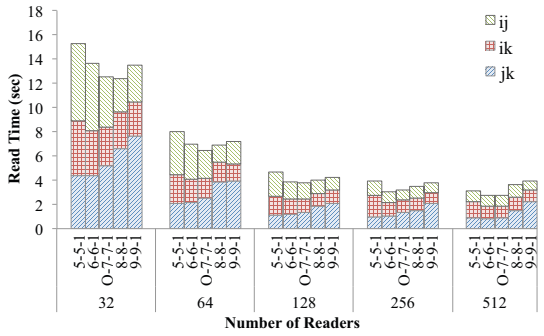


Fig. 8: Dynamic Subchunking Performance; O-7-7-1 (49 subchunks) is our calculated value using the Optimized Chunking formula. We compare the performance with other variation of chunking from 25 (5-5-1) to 81 (9-9-1) subchunks

As we can see, O-7-7-1 delivers the best read time on the slow dimensions in most cases. Increasing or decreasing 1 subchunk on each dimension, that is 6-6-1 and 8-8-1 in our test case, does not achieve a significant performance improvement. Further tuning of the number of subchunks suggests that more performance degradation can be caused by either data overhead or processing overhead. Even though O-7-7-1 does not give the best performance for overall read time in some cases due to the overhead on the fast dimension *jk*, it is still able to deliver a close-to-optimal performance. As described in III, the overall best performance can be achieved anywhere within the *optimized region*. Therefore, our formula is able to provide a satisfactory result for a close-to-optimal

performance, if not the optimal.

C. Planar Read with Hierarchical Spatial Aggregation

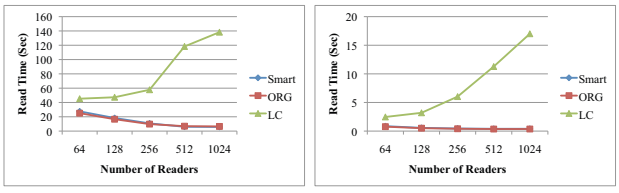
We then examine the performance of Hierarchical Spatial Aggregation by a given OCS. In our experiment, we focus on examining the threshold for performing HSA and its performance impact on planar reads. The calculated lower bound 1.25MB causes a one-level HSA (represented by 1-level in Figure 9 for Var4 with an aggregated chunk size of 2MB. To evaluate the performance of HSA, we manually set the HSA threshold to 4MB and 256KB, which lead to two-level (2-level) of HSA and no HSA (no-aggr) respectively. The results are shown in Figure 9.

Fig. 9: Planar Read Performance of HSA

As we can see, 1-level of HSA provides the best performance on the system. The performance of no-aggr suffers from frequent seek operations while 16MB aggregated chunk size of 2-level HSA introduces large data overhead. 1.25MB threshold of Smart-IO achieves a good balance between the number of seek operations and the amount of redundant data retrieval on the slow dimension. The read performance becomes closer with larger number of processes because the amount of overhead is reduced per process.

D. Read Subvolume

Finally we evaluate the performance of reading a subvolume from a variable. For these experiments, we use variables Var1 and Var4 as the representative cases. A volume 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 global dimension size. Figure 10 shows the experimental results for variables Var1 and Var4.



(a) Var1 (Subvolume = 64GB) (b) Var4 (Subvolume = 128MB)

Fig. 10: Subvolume Performance

As shown in Figure 10, Smart-IO and the original ADIOS achieve comparable performance as the dimension dependency is less obvious for subvolume read access. The performance of LC suffers because data within a subvolume after 3-D domain decomposition is not contiguous, causing significant performance degradation in both cases.

VI. Related Work

Improving I/O performance on large scale systems has been an active research topic in HPC. While much efforts have been focused on write side of issue [34], [35]. Read performance has gained more attention lately. [20] and [15] evaluated and discussed the performance of many of the reading patterns for extreme scale science applications. A number of studies [13], [8], [36], [32] 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.

A line of work has studied the efficient data reorganization for multidimensional data organization. For example, log-based data organization is exploited for databases [12] and various file systems [22], [27], [33]. Sarawagi et al. [23] categorized the strategies for efficient organization of large multi-dimensional arrays. Chunking has been commonly recognized as an efficient data layout for multidimensional arrays because of its capability of alleviating dimension dependency [28]. To further speedup Many multidimensional array declustering algorithms [19], [21], [5], [6] were proposed to improve common access patterns of a multidimensional array.

Schlosser et al. [25] explored the chunk placement strategy at the disk level. [10] and [11] examined different caching algorithm for chunking. However, the future access pattern for scientific application varies and may not be known *a priori*. In [23], Sarawagi et al. gave an initial guidance of what is the proper way for chunking. Sawires et al. [24] and Sorouch et al. [26] proposed multilevel chunking strategies to further improve the performance for range queries on a multidimensional array. Otoo et al. [18] mathematically calculated the optimal size of subchunks from a group of system parameters. However, the study was based on very limited resource, and did not reflect the reality on modern petascale systems.

VII. Conclusions

The complexity of gigantic scientific data, and the physical limitation of current storage system pose a grand challenge on providing an efficient I/O method. This work addresses the read side of I/O issues and focuses on providing a system-aware data organization that can effectively utilize the underlying storage system. To guide the proper data organization for multidimensional scientific data, we have developed the Optimized Chunking model to find the best balance between the data transfer efficiency and the processing overhead. Our model takes a collection of system parameters into consideration, so that the data chunk size can adapt to the underlying system. To further enable such model for scientific applications, we have designed a light-weighted I/O framework named Smart-IO. Smart-IO provides two levels of data organization to address the challenges from a single

storage target and the overall parallel storage system through the Optimized Chunking model and a Hilbert Space Filling Curve ordering. By applying such two-level data organization, we significantly alleviate the dimension dependency for multidimensional scientific data. A much more balanced and consistent read performance is ensured for scientific post-processing. We evaluated Smart-IO on Jaguar Supercomputer at ORNL. Our experimental results show that Smart-IO is able to achieve a maximum of 72 times and 22 times speedup to the planar reads of S3D compared to the Logically Contiguous and chunking data layout, respectively.

In the future, we plan to evaluate the Smart-IO on other type of file systems such as GPFS. Enabling the Optimized Chunking-based data indexing and processing techniques will be investigated too.

VIII. Acknowledgment

This work is funded in part by a National Science Foundation award CNS 1059376, by a NASA award NNX11AR20G, and by a UT-Battelle grant (UT-B-4000099498) to Auburn University. This research used resources of the Oak Ridge Leadership Computing Facility, located in the National Center for Computational Sciences at Oak Ridge National Laboratory, which is supported by the Office of Science of the Department of Energy under Contract DE-AC05-00OR22725.

References

[1] Adaptable I/O System. http://www.nccs.gov/user-support/center-projects/adios .

[2] SDSC Gordon: Data-Intensive Supercomputing. http://gordon.sdsc.edu.

[3] TOP 500 Supercomputers. http://www.top500.org/.

[4] 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.

[5] C.-M. Chen, R. Bhatia, and R. Sinha. Multidimensional declustering schemes using golden ratio and kronecker sequences. *Knowledge and Data Engineering, IEEE Transactions on*, 15(3):659 – 670, may-june 2003.

[6] C.-M. Chen and C. T. Cheng. From discrepancy to declustering: near-optimal multidimensional declustering strategies for range queries. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '02, pages 29–38, New York, NY, USA, 2002. ACM.

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

[8] 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.

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

[10] P. M. Deshpande, K. Ramasamy, A. Shukla, and J. F. Naughton. Caching multidimensional queries using chunks. In *Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, SIGMOD '98, pages 259–270, New York, NY, USA, 1998. ACM.

[11] 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.

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

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

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

[15] J. Lofstead, M. Polte, G. Gibson, S. Klasky, K. Schwan, R. Oldfield, M. Wolf, and Q. Liu. Six degrees of scientific data: reading patterns for extreme scale science io. In *Proceedings of the 20th international symposium on High performance distributed computing*, HPDC '11, pages 49–60, New York, NY, USA, 2011. ACM.

[16] 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.

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

[18] E. J. Otoo, D. Rotem, and S. Seshadri. Optimal chunking of large multidimensional arrays for data warehousing. In *Proceedings of the ACM tenth international workshop on Data warehousing and OLAP*, DOLAP '07, pages 25–32, New York, NY, USA, 2007. ACM.

[19] E. J. Otoo, A. Shoshani, and S. won Hwang. Clustering high dimensional massive scientific dataset. In *SSDBM*, pages 147–157, 2001.

[20] 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.

[21] S. Prabhakar, K. Abdel-Ghaffar, D. Agrawal, and A. El Abbadi. Efficient retrieval of multidimensional datasets through parallel i/o. In *High Performance Computing, 1998. HIPC '98. 5th International Conference On*, pages 375 –382, dec 1998.

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

[23] 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.

[24] A. Sawires, N. E. Makky, and K. Ahmed. Multilevel chunking of multidimensional arrays. *Computer Systems and Applications, ACS/IEEE International Conference on*, 0:29–I, 2005.

[25] S. W. Schlosser, J. Schindler, S. Papadomanolakis, M. Shao, A. Ailamaki, C. Faloutsos, and G. R. Ganger. On multidimensional data and modern disks. In *FAST*, 2005.

[26] T. K. Sellis, R. J. Miller, A. Kementsietsidis, and Y. Velegrakis, editors. *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2011, Athens, Greece, June 12-16, 2011*. ACM, 2011.

[27] 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.

[28] T. Shimada, T. Tsuji, and K. Higuchi. A storage scheme for multidimensional data alleviating dimension dependency. In *Digital Information Management, 2008. ICDIM 2008. Third International Conference on*, pages 662 –668, nov. 2008.

[29] Y. Tian. Src: enabling petascale data analysis for scientific applications through data reorganization. In *Proceedings of the international conference on Supercomputing*, ICS '11, pages 375–375, New York, NY, USA, 2011. ACM.

[30] Y. Tian, S. Klasky, H. Abbasi, J. Lofstead, N. P. R. Grout, Q. Liu, Y. Wang, and W. Yu. Edo: Improving read performance for scientific applications through elastic data organization. In *CLUSTER '11: Proceedings of the 2011 IEEE International Conference on Cluster Computing*, Washington, DC, USA, 2011. IEEE Computer Society.

[31] Y. Tian, S. Klasky, W. Yu, H. Abbasi, B. Wang, and N. Podhorszki. A system-aware optimized data organization for efficient scientific analytics. In *Proceedings of the 21st international symposium on High performance distributed computing*, HPDC '12. ACM, 2012.

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

[33] 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.

[34] W. Yu and J. Vetter. ParColl: Partitioned Collective I/O on the Cray XT. In *International Conference on Parallel Processing (ICPP'08)*, Portland, OR, 2008.

[35] W. Yu, J. Vetter, and H. Oral. Performance Characterization and Optimization of Parallel I/O on the Cray XT. In *22nd IEEE International Parallel and Distributed Processing Symposium (IPDPS'08)*, Miami, FL, April 2008.

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