DynaM: Dynamic Multiresolution Data Representation for Large-Scale Scientific Analysis

Yuan Tian[13] Scott Klasky[2] Weikuan Yu[3] Bin Wang[3]
Hasan Abbasi[2] Norbert Podhorszki[2] Ray Grout[4]

University of Tennessee[1]
{*tiany*}@*ornl.gov*

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

Auburn University[3]
{*tianyua,wkyu,bzw0012*}@*auburn.edu*

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

***Abstract*—Fast growing large-scale systems enable scientific applications to run at a much larger scale and accordingly produce gigantic volumes of simulation output. Such data imposes a grand challenge to post-processing tasks such as visualization and data analysis, because these tasks are often performed at a host machine that is remotely located and equipped with much less memory and storage resources. During the simulation runs, it is also desirable for scientists to be able to interactively monitor and steer the progress of simulation. This requires scientific data to be represented in an efficient form for initial exploration and computation steering. In this paper, we propose DynaM a software framework that can represent scientific data in a multiresolution form, and dynamically organize data blocks into an optimized layout for efficient scientific analysis. DynaM supports a convolution-based multiresolution data representation for abstracting scientific data for visualization at a wide spectrum of resolution. To support the efficient generation and retrieval of different data granularities from such representation, a dynamic data organization in DynaM is enabled to cater distinct peculiarities of different size data blocks for efficient and balanced I/O performance. Our experimental results demonstrate that DynaM can efficiently represent large scientific dataset and speed up the visualization of multidimensional scientific data. An up to 29 times speedup is achieved on Jaguar supercomputer at Oak Ridge National Laboratory.**

I. Introduction

Leadership computers have enabled scientific applications to run at a much larger scale and produce more simulation outputs. The dataset is typically in the order of terabytes or even petabytes. On the other hand, post-processing of such output is often performed on a remote host, which has much less resources such as memory and storage. Data analysis often needs to be performed in pieces because of storage and memory limitation, which is very inefficient. In addition, interactive monitoring and steering of simulation has gained a lot of attention. This technique helps scientists to monitor the simulation execution for early error detection, but has high requirement on I/O speed. To meet such demands, how to move data efficiently for scientific analysis and visualization has become a critical challenge. To ensure scientific data can be quickly retrieved and effectively analyzed, two questions need to be answered: (1) how to represent gigantic multidimensional scientific datasets in a reduced form that still allows scientists a faithful judgment of the progress and correctness of the long-running simulation codes? and (2) how to organize such data so that efficient I/O performance can be delivered to applications on large scale systems?

Multiresolution has been a widely adopted technique in image processing to progressively render large images. The basic idea is to capture the characteristics of image pixels into different levels of resolution. The coarsest level has the smallest size and contains the least information. The image size and details increase with higher resolutions. Example includes Google Map, JPEG 2000 Standard, etc. Such strategy suits well for the offline visualization of data on small computer systems. However, multiresolution techniques need to be well studied before it can be applied for the needs of online monitoring of simulation outputs for computation steering.

Data points in a multidimensional scientific dataset carry important information for scientific exploration. It is desirable for such information to be maintained for a fair judgment of simulation quality, even at the coarser granularity and reduced accuracy.

More importantly, a multiresolution data presentation needs to be well supported by underlying storage system to ensure a fast I/O. When dealing with I/O on large scale system, a data organization that matches well with the storage system is the key to the optimal I/O performance. The challenge for a fast data retrieval mainly comes from the discrepancy of the unconventional access pattern of scientific application, and the physical limitation of storage system. As multidimensional array is the common data structure for scientific data, mapping from n-dimensional logical space to 1-D storage space resulting noncontiguous placement of data points on slow-varying (slow) dimensions. An *imbalanced* performance is often observed for queries on a subset of data on different dimensions, such as reading a 2-D plane from a 3-D array. Such phenomenon is often referred as "dimension dependency" [36], where the read performance depends on the dimensions involved in a query, rather than just the data

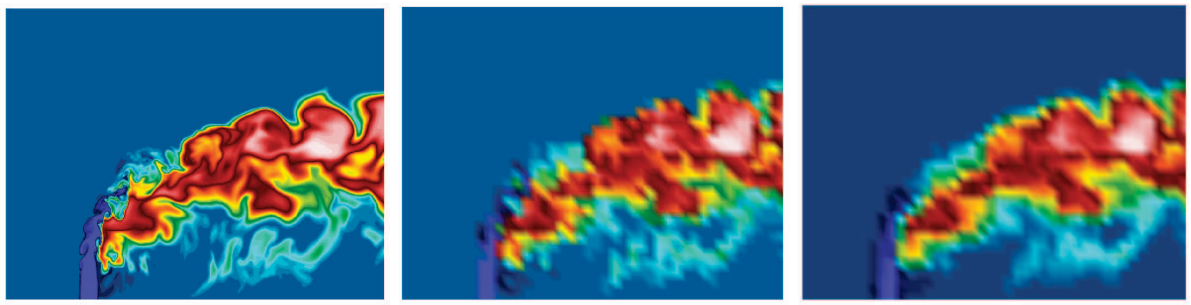

(a) Full Resolution (b) Down-sampling (c) Convolution

Fig. 1: Comparison of Multiresolution Data at a Level-5 Resolution

size of the query.

A collection of research has been carried out for better approaches on efficiently storing and indexing multidimensional array. Among them, a strategy called chunking has become a standard for multidimensional array storage. This approach partitions an array into many smaller units called chunks. While chunking is able to alleviate dimension dependency, it still has constraints in supporting a multiresolutional structure. The most critical challenge is that different data granularities from different level of resolution lead to data blocks of significantly different sizes, which in turns poses a serious challenge for the underlying storage system to support it with good I/O performance: 1) because data size decreases exponentially with the reducing of resolution, a large amount of small data chunks are generated at coarser levels of resolution. Such small data segments in the logical file causes inefficient data retrieval for most access patterns; 2) when data chunk is large, reading on slow dimension involves significant overhead because data points are laid out sequentially within each chunk. Therefore, a data organization strategy that can dynamically cater the storing requirements for different level of resolution is desired.

To this end, we propose **DynaM**, a software architecture that enables **Dyna**mic representation of **M**ultiresolution data. DynaM supports a convolution-based multiresolution data representation for abstracting scientific data for visualization at a wide spectrum of resolution. An Optimized-chunk algorithm [41] supported dynamic data organization is used to cater distinct peculiarities of different size data blocks for efficient and balanced I/O performance.

We have designed and implemented DynaM as a part of the ADaptive I/O System (ADIOS) framework [1]. Our experimental evaluation on the Jaguar supercomputer at Oak Ridge National Laboratory demonstrates that DynaM can present scientific data dynamically at any necessary granularity of resolution. Figure 1 shows an output of a 2-D slice from a DynaM-based S3D [6] simulation, compared to the same slice produced with a conventional down-sampling strategy. With only $\frac{1}{256}$ of the original size, DynaM is able to enable data visualization with better quality. We also observe that DynaM helps enable efficient read operations even for challenging scientific access patterns.

The rest of the paper is organized as follows. We introduce the design of DynaM in Section II. Section III validates our strategy through a comprehensive set of experimental results. A survey of literature review is provided in Section IV. Finally, we conclude the paper in Section V.

II. Design of the DynaM Software Architecture

In view of the needs of multiresolution data representation and accordingly efficient I/O support on large-scale systems, we choose to design DynaM as a part of ADIOS (Adaptable I/O System) [1], an I/O middleware developed by National Laboratory. ADIOS has been demonstrated as an efficient I/O library that provides significant performance benefits for a number of petascale scientific applications [2], [18], [20], [45], [27]. It uses a default file format called BP (Binary Packed). In this format, ADIOS applies the chunking strategy for storing multidimensional datasets. DynaM retains many salient features of ADIOS, while it focuses on multiresolution representation of multidimensional datasets and accelerates data retrieval for data visualization and analysis. Figure 2 shows the software architecture of DynaM.

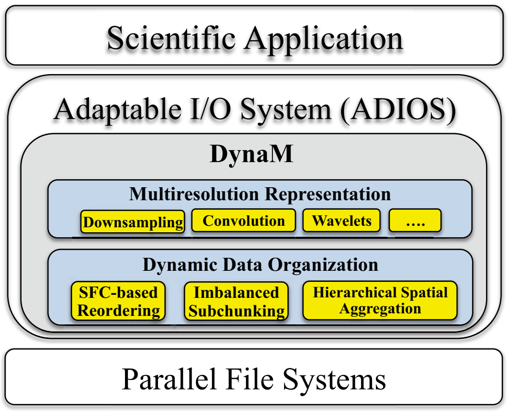


Fig. 2: Software Architecture of DynaM

DynaM is composed of two major components: *Multiresolution Representation* and *Dynamic Data Organization*. The multiresolution component focuses on enabling different multiresolution algorithms for data representation. The dynamic organization component is designed based on our earlier study in [41], [39]. It focuses on enabling different data organization strategies to cater distinct peculiarities of data blocks at each level of resolution under the govern of an *Optimized Chunking* policy. The data organization strategies inside this component are *Hierarchical Spatial Aggregation*, *Imbalanced Subchunking*, as well as the default SFC (Space Filling Curve)-based distribution of data chunks among storage devices.

A. Multiresolution Data Representation

Current multiresolution techniques can be categorized into two categories. The first category of strategies is sampling based, in which data points with different subsampling distances are selected to construct a level of resolution. The finer level consists of more data points to produce the details. Such down-sampling algorithms are normally fast because of its simplicity. The main issue is that some of data points are inevitably lost during down-sampling, which may carry important information. In addition, a discontinuity, a.k.a, error, is commonly found at the boarder between computation processes in scientific simulation. An example is shown in Figure 3, where the blue solid line shows the full resolution carries a anomaly. The red dashed line shows a coarser level of data points through down-sampling of the original data points. with this strategy, the spike in the full resolution is completely missed. A more sophisticated multiresolution technique is needed to achieve a better approximation of the original curve, as shown by the green dotted line. This approximation is achieved through convolution, which is described later in this section. There are many approximation strategies through mathematical transformations, such as Fourier Transform, or Wavelet Transform, etc. Among them, wavelet-based transformation has attracted a lot of attention since its application in JPEG2000 standard. Wavelet-based algorithm is able to produce a good representation of the original data with limited data size combining with compression. The major concern with this kind of technique is that they are computationally expensive, which leads to longer I/O time. While scientific applications running on large-scale systems normally have restricted requirements on I/O.

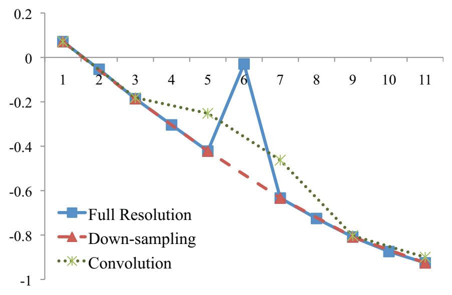


Fig. 3: Data Sampling Strategies

We introduce a multiresolution data representation based on convolution to accommodate the demands of both quality and efficiency in representing scientific data. Convolution is a mathematical calculation that takes two function f and g, produces a third function that is typically viewed as similar to one of the original functions. It has been widely used on many applications such as signal and image processing, etc. The fundamental idea is that each pixel of an image is updated by a new value that is calculated based on itself and all the immediate neighboring pixels. The contribution of each pixel in the calculation is defined by the convolution kernel used. Different effects can be achieved by using different convolution kernels. Common convolution kernel include as Gaussian kernel, mean kernel, Laplacian kernel, etc. Kernels can also be customized by users for special purpose. Figure 4 gives an example of 2D convolution using mean kernel. By taking an average of itself with the neighboring 8 pixels, the value in the middle is updated to 7.

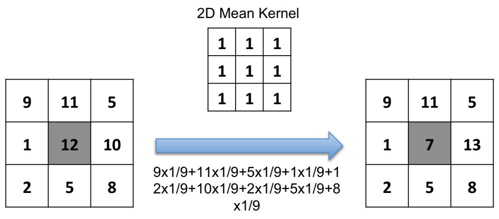
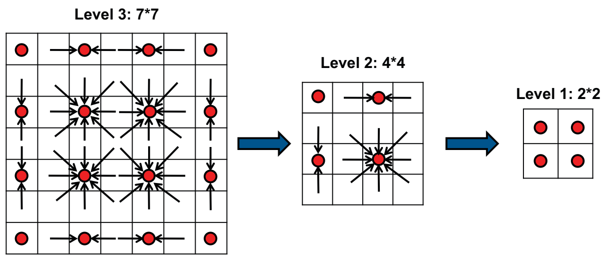


Fig. 4: 2D Convolution Using Mean Kernel

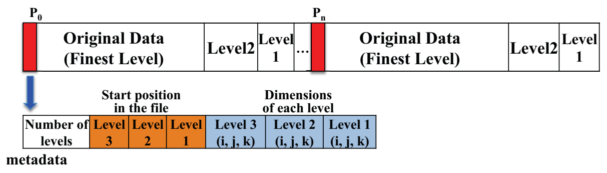
Convolution demonstrates the possibility of using one value to represent the characteristics of its neighboring region. Thus we adopt this idea for our multiresolution representation of scientific data. The only issue is the process of border points need to be specifically treated. Conventional convolution does not have specified operation for border points. In image processing, it is normally handled by padding the border region using adjacent values to construct a region which is operable by convolution kernel. Such operation involves more memory access and calculation. Thus in our algorithm, we apply 1D, 2D or 3D convolution for different types of data points whichever is possible.

In Figure 5(a), we show an example of how two levels of resolution are generated from the full resolution (Level-3) for a 2D 7*7 array. According to our algorithm, the corner points keep their original values. A 1D convolution is performed for all edge points; 2D convolution is performed for the rest of data points. Except one coarse level (level 2) is calculated using the original data points in full resolution, the rest of coarse levels, Level-1 in our example, are calculated based on the previous level of resolution. The reason of this recursive design is to limit the overhead to data generation. Because data points at lower resolution are sparsely located, which requires a larger convolution kernel to cover the intermediate region. And the complexity of calculation increases with the increasing size of convolution kernel size. By applying convolution repeatedly on the previous level of resolution, each level can be quickly generated without causing significant overhead for write. Meanwhile, the characteristics of the original data is able to be passed along even to the coarsest level of resolution.

In the chunking organization, each process is assigned a data chunk after domain decomposition. Each process performs convolution on its own data chunk, and stores each level of



(a) Convolution of a 2D Chunk



(b) File Format and Metadata Organization

Fig. 5: Generation of Multiresolution and Output

resolution as an increment to its original data, as shown in Figure 5(b). The corresponding metadata is also generated at each process to identify different levels of resolution. From a global view, the combination of one level of resolution from all the processes naturally constructs the correspondent level of resolution for the entire data space. A subset of data can be dynamically constructed as well. It also maintains the benefit of using chunking layout for storing multidimensional arrays. Since storage is getting larger and cheaper quickly, storing redundant data becomes quite common on large-scale systems. Many studies have shown that, by storing a little more, much performance gain can be achieved, such as [3], which has been successfully demonstrated by ADIOS [23] itself. In fact, the data overhead is also very limited in such design. With a decreasing level of resolution, the data size reduces down to $\frac{1}{2^n}$ of the previous level, where n is the number of dimensions. For a 3D array with five level of resolution, the amount of data overhead is about 14%, which does not necessarily reflect the same level of write performance degradation. The coarsest level of resolution only has the size of $\frac{1}{4096}$ of the original data. A much faster read can be expected.

B. Dynamic Data Organization

Multiresolution representation allows visualization and analysis at different granularities for scientific datasets. With multiresolution data, scientists can decrease the amount of data movement for coarser analysis, thereby increasing the throughput and productivity of data analysis. However, fast data visualization and analysis is not guaranteed on large-scale systems without a data organization that matches well with the underlying storage subsystem. Thus, we adopt the dynamic data organization based on our earlier work [41], [39] to ensure efficient I/O for multiresolution datasets. The crux of such strategy is an *Optimized Chunking Size* (OCS) [41], which is defined as $OCS = BW_{io} \times (CC + T_s)$, where $BW_{i/o}$ is the I/O bandwidth of one storage node, T_s is the time unit for each seek operation, and CC is the communication cost unit. The detail of this algorithm can be found in [41]. OCS denotes the size of data chunks that can achieve a good balance between data transfer efficiency and processing overhead. On top of that, *Hierarchical Spatial Aggregation (HSA)* and *Imbalanced Subchunking (SUB)* are used to organize data into a size that is close to OCS. A high-level description is provided here. For a n-dimensional array, the organization policy can be described as:

- If chunk size is smaller than OCS, use Hierarchical Spatial Aggregation to consolidate small chunks;
- If chunk size is larger than OCS, use Imbalanced Subchunking to decompose large chunks;
- If chunk size satisfies defined OCS, use Space Filling Curve for chunk-level reordering.

Under such policy, a data chunk at each process has three paths to be stored on the file system as shown in Figure 6. Below we present the design of multiresolutional data representation with such dynamic data organization. The detail of algorithms can be found in [41].

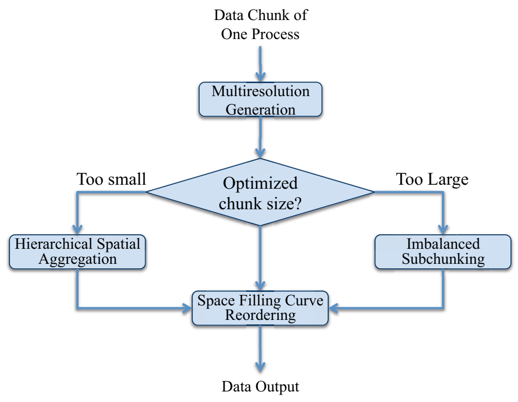
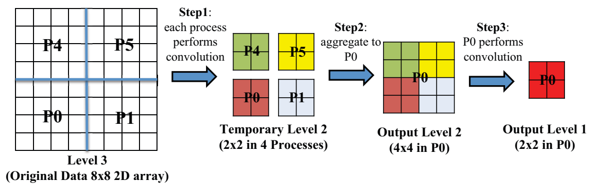

Data Chunk of One Process
Multiresolution Generation
Optimized chunk size?
Too small
Too Large
Hierarchical Spatial Aggregation
Imbalanced Subchunking
Space Filling Curve Reordering
Data Output

Fig. 6: Flow of Multidimensional Data with Dynamic Data Organization

1) Multiresolution with Hierarchical Spatial Aggregation: By using multiresolution technique, the data size decreases exponentially with the decreasing of resolution. At the coarsest levels of resolution, a significant amount of small data blocks are generated. For example, if every chunk has 64MB of data with 4 level of resolution (full resolution is included), the coarsest level is only 128KB. For an application running with 8,000 processes, the entire data space is divided into 8,000 small segments distributing within the output file. A significant amount of seeks and memory operations are required for common access patterns such as planar reads, correspondingly leads to limited read performance. Aggregation is commonly used to converge small pieces of data. However, simply concatenating small chunks does not solve the problem this case. Because the number of disk and memory operations remains the same for reading. Thus, we designed a Hierarchical Spatial Aggregation strategy which aggregates data chunks in a way that their spatial localities are reserved. Figure 7(a) shows such an example of aggregating from 4 processes in a 2D space.

Each process first generates first performs convolution to generate its first level of resolution data. Then for every spatially adjacent 4 process, one process is selected as the aggregator. It aggregates data from neighboring 3 client processes and construct a larger data chunk. If 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 aggregator. After aggregation, because an aggregator has all the data of neighboring processes, there is no need for all the clients to perform convolution on its data for the next level of resolution. Otherwise, another round of aggregation may be invoked since data size may fall below the optimized chunk size. Therefore, only the aggregator performs convolution on aggregated data to generate the next level of resolution. Even though such strategy leads to more computation at the aggregator, the computation is much cheaper than memory operation on high end computers. If the generated data size is below the optimized chunk size, another round of HSA will be performed but only among aggregators.



P4
P5
P0
P1
Level 3
(Original Data 8x8 2D array)
Step1: each process performs convolution
Temporary Level 2
(2x2 in 4 Processes)
Step2: aggregate to P0
Output Level 2
(4x4 in P0)
Step3: P0 performs convolution
Output Level 1
(2x2 in P0)

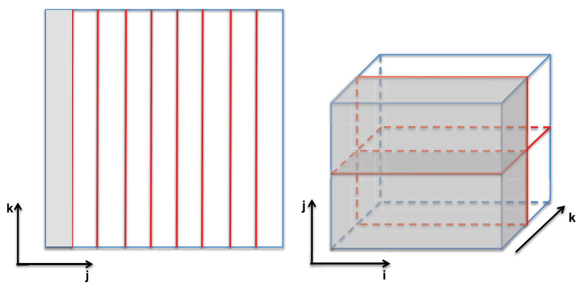
(a) Spatial Aggregation

Fig. 7: Hierarchical Spatial Aggregation

To avoid over aggregating data chunks, only the chunk size is less than $\frac{OCS}{2^n}$ will be aggregated. For example, in a 3-D space, a minimum of aggregation is performed on 7 neighboring processes, leading to an upper bound chunk size $\frac{OCS}{8}$.

2) Imbalanced Subchunking: Chunking has shown its capability to alleviate the *dimension dependency* problem for reading. However, because data points are laid out sequentially within a chunk, dimension dependency can be significant again when a data chunk is large. With chunking, a common practice for a range query on a slow dimension is to read in a lot of redundant data from the start to the end point of request to avoid the frequent seek operations among noncontiguous data points. Even though such strategy is more preferable than frequent seeks on parallel file system, its performance can suffer when the chunk is too large, meaning significant amount of redundant data to retrieve. As optimized chunk size 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 OCS to further alleviate dimension dependency. Chunking large arrays is important when efficient I/O is desired in applications that access data with a high degree of locality.

To maximize the reduction for redundant data retrieval on the slowest dimension, we enable an imbalanced subchunking based on our early study in [41]. Such strategy performs a domain decomposition on all the dimensions except the slowest dimension. Figure 8 shows an example of such strategy for a 2D and a 3D data chunk, respectively. The shaded region represents the amount of data needs to be read in for a request on the slowest dimension. As we can see, only a fraction of data are retrieved after subchunking instead of the entire data chunk with the original data layout. However, the performance on the fast dimension is expected to degrade due to the break of contiguity. The rational behind sacrificing the performance on the fast dimension to compensate the slow dimension(s) is that reading the contiguous data is normally very fast on high-end systems. For example, reading in 200MB contiguous data on Jaguar supercomputer only takes less than 1 second. Introducing a limited number of seeks into fast dimension does not significantly hurt the user experience, but improve the read performance on slow dimension(s) dramatically.



(a) 2D (b) 3D

Fig. 8: Imbalanced Subchunking

3) 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, as shown in Figure 9. The rational of such strategy is based on our earlier work [40], in which we studied how the read performance on a subset of data from a multidimensional array can be impacted by access patterns and data placement. Because scientific applications have unconventional access patterns, chunking based data organization faces a reduced read concurrency issue when data chunks are placed sequentially on storage targets. When a subset of data elements is requested, the data may be concentrated on one or few storage targets even if they are not logically contiguous in the data array. Thus reading such data can not employ the aggregated bandwidth from all storage devices. This leads to a phenomenon called *imbalanced performance* across dimensions. In DynaM, we inherit the Hilbert Space Filling Curve [13] based ordering from our earlier work and rearrange the placement of optimized chunks on storage devices. By doing so, data concurrency is no longer constrained by the access pattern. Reading on any dimension can leverage the close-to-maximum aggregated bandwidth.

III. Experimental Results

We have implemented and evaluated DynaM on the *Jaguar* supercomputer at Oak Ridge National Laboratory (ORNL). Jaguar is currently the third fastest supercomputer in the world [22]. Jaguar is equipped with Spider (an installation

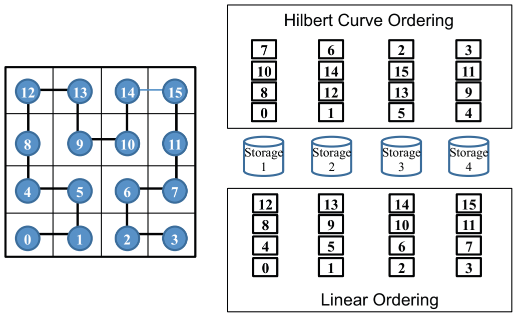


Fig. 9: Comparison of Linear Placement and Hilbert Curve Reordering of 16 Chunks on 4 Storage Nodes

of Lustre) for the storage subsystem, with a demonstrated bandwidth of 240 GB/s. Spider has three partitions named Widow 1, Widow 2 and Widow 3, respectively. Each partition contains 336 storage targets (OSTs). In our experiments, we used its Widow 2 partition.

S3D [6] combustion simulation code 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 space. Two representative test cases: small (S), large (L) and their chunk size (Elements/Data Size) are shown in Table I with level of resolutions we used in our evaluation. Three levels of resolution are generated for Case S and four levels for Case L. With these settings, the chunk size is reduced to 32KB and 256KB respectively at the coarsest level, Level-1. The I/O bandwidth of Spider is approximately 250MB/Sec per OST, the average seek time is 8ms, and the communication cost is about 19ms. Thus, the OCS is calculated as 2.5MB using our Optimized Chunking algorithm [41]. Based on the previous practices of the ADIOS team at Jaguar, the stripe size is set as the size of ADIOS process group. This can maximize data concurrency and reduce false sharing on the Lustre file system, and reduces internal and external interferences [19].

TABLE I: Test Cases and Resolution

	Level-4 (Original Data)	Level-3	Level-2	Level-1
S		64^3/2MB	32^3/256KB	16^3/32KB
L	256^3/128MB	$128^3$16MB	64^3/2MB	32^3/256KB

The performance evaluation of DynaM is mainly focused on the data representation and I/O performance. Planar read is mostly used in I/O test cases as it is a common yet very challenging access pattern. We measure the read performance between three types of data organization strategies: Logically Contiguous (LC), the original ADIOS with Hilbert curve ordering at the chunk-level (ORG), and DynaM(DM). 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 Representation of Multiresolution

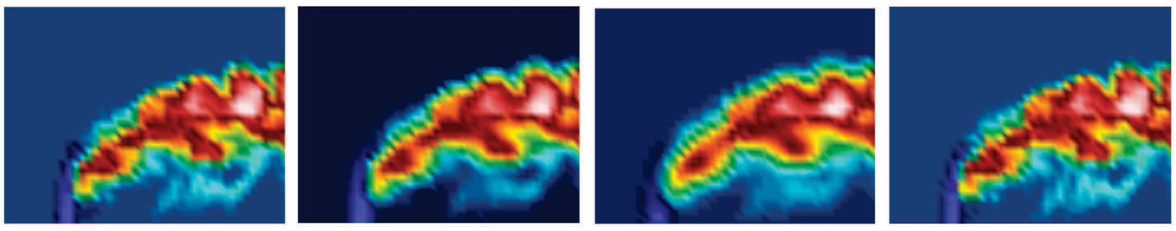
We apply both convolution and down-sampling on a real S3D simulation output and compare the visualization image of a 2-D slice from a 3-D data array. The results are shown in Figure 1. A mean kernel is used for the convolution of DynaM in this case. As we can see, most of the boarders is diminished when using down-sampling. Even though the data generated by S3D is smooth in this case, DynaM still has relatively accurate representation of the original image by using only $\frac{1}{256}$ of the original data.

In image processing, different convolution kernels can be used for different results of output. There are common convolution kernels such as Mean kernel, Gaussian kernel, Laplacian kernel, etc. These kernels can be further customized by users for special purposes. As an example, we compare the output at Level-1 resolution by using the mean kernel with sizes of 3, 5, 7 and the Gaussian kernel with the size 3. The numbers represent the number of elements participating in the calculation on each dimension. The results are shown in Figure 10. The Mean kernel produces clearer presentation of the original image than The Gaussian kernel with the same size of kernel. This is expected because the Gaussian kernel is normally used to add blurring to an image. Because a bigger kernel size means that more neighboring points are gathered, correspondingly we find a more abstract output of the original image with larger size of convolution kernels. Note that different application may have different requirements and data characteristics. One type of convolution kernels does not guarantee the best output for all the applications. DynaM allows user to specify the type and the size of convolution kernels with a flexible interface.

B. Data Generation

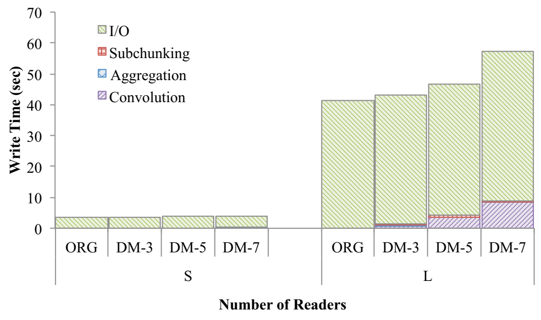
One of the design considerations of DynaM is to contain the performance impact of data generation. To evaluate the data generation overhead, three levels of resolution are generated for Case S and four levels for Case L by using convolution. Based on 2.5MB OCS size, a two-level HSA will be performed starting from level-2 resolution for Case S, and a one-level HSA at level-1 for Case L. Moreover, Imbalanced Subchunking is applied to its original data in Case L. The cost of convolution is directly impacted by convolution kernel size. We also include this parameter into evaluation. The write time of using 4,096 processes along with the time breakdown is shown in Figure 11(a). The number following DM represents the convolution kernel size.

As we can see, the increase of the total write time is mostly determined by the convolution kernel size. Only limited write overhead is introduced for both cases until the kernel size reaches 7 in Case L. In this case, every convolution operates on a 7^3 3-D volume, which is computational expensive and has deeper impact when the data size is large. Other operations such as subchunking, aggregation do not cause significant delays to the write time. We then fix the convolution kernel size at 3 and evaluate the weak scaling of DynaM. As shown in Figure 11(b) and 11(c), very limited overhead is brought

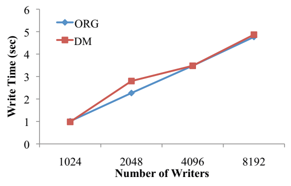


(a) Mean Size=3 (b) Mean Size=5 (c) Mean Size=7 (d) Gaussian Size=3

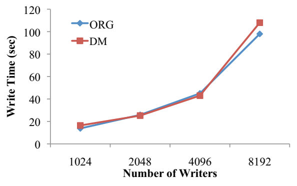
Fig. 10: Comparison of Different Convolution Kernels on the Coarsest Resolution



(a) Write Time Breakdown



(b) Weak Scaling of Case S



(c) Weak Scaling of Case L

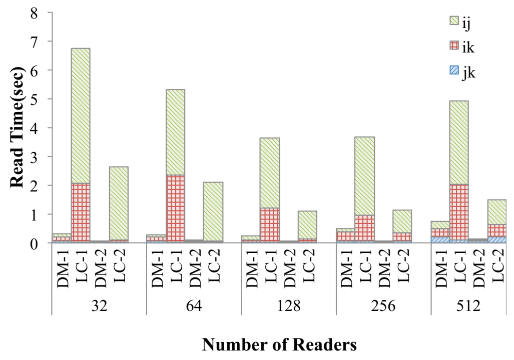
Fig. 11: Data Generation Overhead

to data generation in all cases. In the following experiments, we fix the convolution kernel size at 3. The data is always generated with 3 levels or resolution for Case S, and 4 level of resolution for Case L.

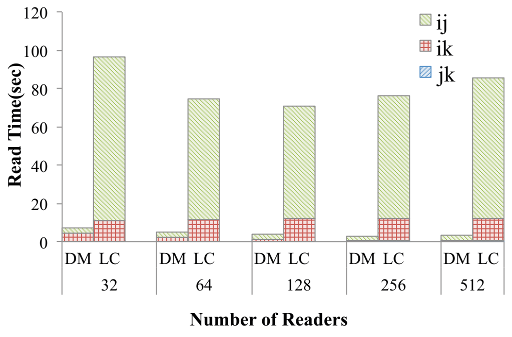
C. *Planar Read Performance of DynaM*

The evaluation of multiresolution data retrieval is mainly focused on the comparison of data organization between DynaM, which uses chunking, and logically contiguous (LC). Logically contiguous layout stores each level of resolution as a contiguous segment within the logical file. A 2-D plane is read from each dimension of a 3-D variable (k is the fastest 281 dimension, and i is the slowest dimension), a common practice in scientific analysis. The number of readers varies from 32 to 512, so to follow the tradition that application scientist often spend only 10% of the writers to read. The total read time of planar reads on three dimensions for both Case S and L are shown in Figure 12. Note the number following the name of each data organization represents the level of resolution.

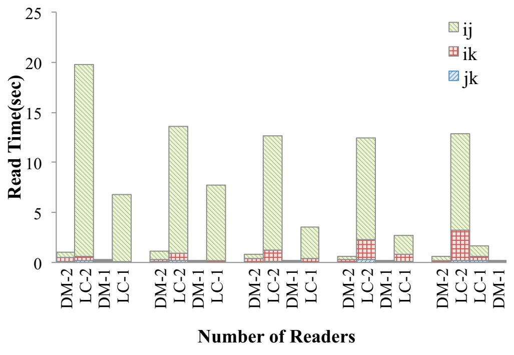
As shown the figure, LC demonstrates a severe imbalanced performance is observed across dimensions in all test cases. While it achieves fast read on the fastest dimension, the jk dimension, the performance of LC suffers either the frequent



(a) Case S



(b) Case L:Level-3



(c) Case L:Level-2 and Level-1

Fig. 12: Planar Read Performance on Different Resolution (1 is the coarsest resolution)

seek operations, or lack of concurrency from the storage system. DynaM delivers a consistently good performance in all cases by using its dynamic data organization. At the coarse levels, such as Level-1 and -2 of case S and Level-1 of Case L, the Hierarchical Spatial Aggregation is able decrease the seek operations between small chunks to speedup read. At the finer level, such as Level-3 of Caes L, Imbalanced Subchunking is able provide an efficient read performance by reducing the redundant data retrieval. The concurrency of the underlying storage system is also well utilized by using Hilbert Curve ordering for chunk placement, so that close-to-maximum aggregated bandwidth is guaranteed from storage. Together, much improved and balanced performance is observed using DynaM compared to LC. An up to 29 times speedup is achieved.

D. Resolution Rendering Performance of DynaM

Finally we evaluate the resolution rendering performance of DynaM. Table II shows the time to read the entire variable at the coarsest level for Case S and Case L by using the number of readers from 64 to 1,024. With the reduced data size, the entire variable can be read out by DynaM within 1 second for both cases, a maximum of 17 times faster compared to the peak performance of LC. At the same time, DynaM demonstrates good scalability due to its flexibility of data organization using any of the three possible paths.

TABLE II: Read Time of The Coarsest Level Using Different Number of Processes (Sec)

		64	128	256	512	1024
S	DM	0.54	0.25	0.27	0.26	0.32
	LC	1.43	1.22	1.57	3.36	5.49
L	DM	0.58	0.30	0.15	0.22	0.17
	LC	3.39	2.51	6.16	11.50	15.97

IV. Related Work

Multiresolution techniques have been widely studied in image processing and visualization domain. The fundamental idea is to abstract a much larger dataset with into limited size of data. Among all the algorithms, Wavelet [7] is a technique to abstract the signal of an image or a dataset into a small subregion and store the filtered signal residuals in the other hierarchical regions, leaving the space for high compression ratio. It was first introduced by Mallat in [21]. Since its successful application in JPEG2000 [38], [37], Wavelet has been widely used in large image and large volume data compression and visualization. Poulakidas et al [28] proposed a wavelet based image compression algorithm for fast image subregion retrieval. Ihm [14] et al introduced a 3-D wavelet algorithm for large volume data rendering and achieved a very good compression ratio. In [12], Guthe et al investigated interactively rendering of very large data sets on standard PC hardware. The data was compressed using a hierarchical Wavelet Transform and decompressed on-the-flay while rendering. Yu et al [44] et al proposed a wavelet-based time-space partition (WTSP) tree algorithm to achieve high compression ratio for large datasets on time domain. [8] proposed a toolkit named VAPOR which applied wavelet transform on scientific data for multiresolutional purpose. In general, most of the wavelet-based algorithms require a preprocessing step for data preparation and are computationally expensive. For scientific applications which have restrict I/O performance requirement, data representation is not the only concern. Pascucci et al proposed IDX [26] for visualization of scientific data through downsampling along Z-order Space Filling Curve. They showed good read and write performance [15]. However, the issue with this work along with other downsampling-based algorithms is, the points of interest may be lost during the subsampling.

On the other side, improving I/O performance on large scale systems has been one of the major research topics in HPC [42], [17], [16]. Many researches have been focused on achieving such a goal by investigating different data organizations for multidimensional scientific data. Same interest is shared in other domains, particularly in database domain, where multidimensional array is commonly used for data storage. For example, log-based data organization is exploited for databases [11] and various file systems [30], [35], [43]. Sarawagi et al. [31] categorized the strategies for efficient organization of large multidimensional arrays into four classes, namely chunking, reordering, redundancy, and partitioning. While chunking has been commonly recognized as an efficient data layout for multidimensional arrays because of its capability of alleviating dimension dependency [36]. A line of work indicated in order to ensure good I/O performance, reorganizing data chunks is necessary. Many multidimensional array declustering algorithms [25], [29], [4], [5] were proposed to improve common access patterns of a multidimensional array. Our previous work [40] applied similar approach and showed an improved read performance for scientific applications running on large scale systems. Schlosser et al [33] explored the chunk placement strategy at the disk level. Much more efficient access was observed by putting data chunks on the adjacent disk blocks. Deshpande et al [9] examined how chunking the large dataset and caching the chunks with a modified LRU algorithm can speedup reading. However, the future access pattern for scientific application varies and may not be known *a priori*. A data layout should accommodate any access pattern. Fan et al. [10] proposed a latin cube strategy to put neighboring elements into one shared memory module to improve I/O performance. Chunking can be further combined with partitioning, a.k.a *subchunking*, which further decomposes the data chunk. In [31], Sarawagi and Stonebraker gave an initial guidance of what the proper way for chunking would be. Then Sawires et al [32] proposed a multilevel chunking strategy to further improve the performance for range queries on a multidimensional array. Otoo et al [24] mathematically calculated the optimal size of subchunks from a combination of system parameters. However, the study was based on very limited resource, and did not reflect the reality on modern petascale systems. A group of researchers proposed SciDB [3] as a data management system particularly intended for applications involving large scale array processing. They used an overlapping chunks to speed up queries on the subset of data. The performance was gained by more requirement for storage without introducing more complexity to applications. In [34], Sorouch et al proposed ArrayStore, a two-level regular and irregular data chunking layout to speedup queries in database system.

In this work, we show a design of elastic data layout that supports multidimensional data presentation so that large data volume can be presented with limited data size. The performance of such layout is ensured through an Optimized Chunking strategy.

V. Conclusion

We have designed and developed DynaM, a software framework that can represent scientific data in a multiresolution form and organize data blocks efficiently as a chimera of three distinct strategies. DynaM provides a flexible interface for supporting different multiresolution algorithms for scientific data representation. Particularly, we introduce a new algorithm which applies the convolution function to a reduced number of data points. So scientific data can be well represented even with a reduced data size. To be able to retrieve the multiresolutional data efficient for scientific analysis, we presented a dynamic data organization which is built upon our earlier work in data organization [41], [40]. Governed under an Optimized Chunking policy, the Hierarchical Spatial Aggregation and Imbalanced Subchunking strategies construct the multiresolution data into chunks suitable to exploit the best I/O performance of underlying file system. In the end, data chunks are placed on the storage through Space Filling Curve ordering to achieve close-to-maximum data concurrency.

We evaluate DynaM on Jaguar Supercomputer at Oak Ridge National Laboratory. It is able to generate a satisfactory representation of the original data even at the coarsest level of resolution. Our experimental results demonstrate that DynaM-based visualization of S3D simulation data can achieve a good rendering of the original dataset. Moreover, the read performance can be improved by as much as 29 times compared to the original logically contiguous data organization.

One future direction will be studying the different convolution kernel for different applications. Eventually we want to build an adaptive multiresolution framework which is able to use different algorithms on data based its own characteristic. Another ongoing research effort is in examining the combination of the multi-precision and multi-resolution for further data reduction for efficient scientific data analysis.

VI. Acknowledgment

This work was funded in part by a UT-Battelle grant to Auburn University and by National Science Foundation award CNS-1059376. This research used resources of the National Center for Computational Sciences at Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

References

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

[2] 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.

[3] P. G. Brown. Overview of scidb: large scale array storage, processing and analysis. In *Proceedings of the 2010 international conference on Management of data*, SIGMOD '10, pages 963–968, New York, NY, USA, 2010. ACM.

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

[5] 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.

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

[7] C. K. Chui. An introduction to wavelets. 1992.

[8] J. Clyne. The multiresolution toolkit: Progressive access for regular gridded data. In *Proceedings of the Visualization, Imaging, and Image Processing*, 2003.

[9] 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.

[10] 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.

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

[12] S. Guthe, M. Wand, J. Gonser, and W. Straßer. Interactive rendering of large volume data sets. In *IEEE Visualization*, 2002.

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

[14] I. Ihm and S. Park. Wavelet-based 3d compression scheme for very large volume data. In *In Proceedings of Graphics Interface '98*, pages 107–116, 1998.

[15] S. Kumar, V. Vishwanath, P. Carns, B. Summa, G. Scorzelli, V. Pascucci, R. Ross, J. Chen, H. Kolla, and R. Grout. Pidx: Efficient parallel i/o for multi-resolution multi-dimensional scientific datasets. In *CLUSTER '11: Proceedings of the 2004 IEEE International Conference on Cluster Computing*, Washington, DC, USA, 2011. IEEE Computer Society.

[16] Z. Liu, B. Wang, P. Carpenter, D. Li, J. S. Vetter, and W. Yu. PCM-based durable write cache for fast disk I/O. In *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2012 IEEE 20th International Symposium on*, pages 451–458. IEEE, 2012.

[17] Z. Liu, B. Wang, T. Wang, Y. Tian, C. Xu, Y. Wang, W. Yu, C. Cruz, S. Zhou, T. Clune, and S. Klasky. Profiling and improving I/O performance of a large-scale climate scientific application. In *ICCCN*. IEEE, 2013.

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

[19] 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.

[20] 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.

[21] S. Mallat. A theory for multiresolution signal decomposition: the wavelet representation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 11(7):674 –693, jul 1989.

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

[23] Oak Ridge National Laboratories. *ADIOS 1.0 User's Manual*, Nov 2009.

[24] 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.

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

[26] V. Pascucci and R. J. Frank. Global static indexing for real-time exploration of very large regular grids. In *SC*, page 2, 2001.

[27] 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.

[28] A. S. Poulakidas, A. Srinivasan, Ö. Egecioglu, O. H. Ibarra, and T. Yang. Image compression for fast wavelet-based subregion retrieval. *Theor. Comput. Sci.*, 240(2):447–469, 2000.

[29] 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.

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

[31] 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.

[32] 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.

[33] 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.

[34] 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.

[35] 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.

[36] 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.

[37] A. Skodras, C. Christopoulos, and T. Ebrahimi. The jpeg 2000 still image compression standard. *Signal Processing Magazine, IEEE*, 18(5):36 –58, sep 2001.

[38] D. Taubman and M. Marcellin. Jpeg2000: standard for interactive imaging. *Proceedings of the IEEE*, 90(8):1336 – 1357, aug 2002.

[39] Y. Tian. Smart-io: System-aware two-level data organization for efcient scientic analytics, 2012.

[40] 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.

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

[42] Y. Tian, Z. Liu, S. Klasky, B. Wang, H. Abbasi, S. Zhou, N. Podhorszki, T. Clune, J. Logan, and W. Yu. A lightweight I/O scheme to facilitate spatial and temporal queries of scientific data analytics. In *Mass Storage Systems and Technologies (MSST), 2013 IEEE 29th Symposium on*. IEEE, 2013.

[43] 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.

[44] H. Yu, C. Wang, R. W. Grout, J. H. Chen, and K.-L. Ma. In situ visualization for large-scale combustion simulations. *IEEE Computer Graphics and Applications*, 30(3):45–57, 2010.

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