Smart-IO: System-Aware Two-Level Data Organization for Efficient Scientific Analytics

Yuan Tian
Advisor: Weikuan Yu
Auburn University
tianyua@auburn.edu
Submitted to the ACM Student Research Competition Grand Finals 2012

***Abstract*—For the last five years, computing power has grown at an unprecedented rate and it is projected that High End Computing systems reach exascale by 2018. These systems enable application scientist to simulate their science with great complexities and consequently, produce a large volume of data with highly complex organizations. Such data imposes a grand challenge to conventional storage systems for the need of efficient I/O solutions during both the simulation runtime and data post-processing phases. Current I/O techniques that have pushed the write performance close to the system peak usually overlook the read side of the problem. With the mounting needs of scientific discovery, the read performance for the large-scale scientific application actually has becomes a critical issue for the HPC community, In this study, we propose SMART-IO, a system-aware two-level data organization framework that can organize data blocks of multidimensional scientific data efficiently through three elastic strategies under the governance of an Optimized Chunking model. This framework can dynamically adapt data organization based on simulation output and underlying storage systems, thereby enabling efficient scientific analytics. Our experimental results demonstrate that Smart-IO achieves up to 72 times speedup for mission critical combustion simulation S3D, compared to the logically contiguous data layout.**

I. Introduction

The increasing growth of leadership computing capabilities, in terms of both system complexity and computational power, has enabled scientific applications to solve complex scientific problems at large scale. Domain scientists, such as high energy physics, climate, chemistry, etc, are leveraging large-scale systems for intensive scientific investigation, pushing the progressing of the human society. Such phenomenon is accompanied by a gigantic volume of complex scientific data produced, driving the impetus for data intensive computing as a very significant factor in scientific computing.

Many efforts, both past and present, have heavily focused on improving the I/O performance by studying the output 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 and, as a result, have a substantial negative impact on the read performance. [8] reported that nearly 90% of time was spent on I/O during the visualization workflows. The major reason is the discrepancy between the physical limitations of magnetic storage and the common access patterns of scientific applications. Physical disks in most HPC systems are optimized for one-dimensional large sequential blocks of data while scientific datasets are normally 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 (fast dimension). This results in noncontiguous placement of data elements on secondary and tertiary dimensions (slow dimensions), as shown in Figure 1. When reading the data elements in the order of higher dimensions, the performance degrades significantly due to the expense of coping with noncontiguity with extra disk seeks and retrieving extra data between needed segments. 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. Without optimizing data organizations, simply using new hardware such as SSDs cannot maximize concurrency and therefore peak bandwidth.

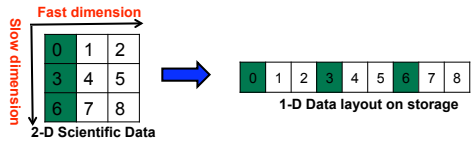


Fig. 1: Multidimensional Data and its Disk Organization

The complexity of the scientific data imposes another challenge on efficient I/O. One simulation output is normally a collection of many multidimensional variables with distinct characteristics. For example, one variable can be 100GB while another variable is only 10MB. Such distinct variables pose different I/O challenges. It is difficult to achieve the optimal performance by applying a uniform data organization strategy. Therefore, a technique that can dynamically organize each variable to match with the storage system is desired.

To address the aforementioned issues, we propose a two-level data organization scheme named Smart-IO. The first level focuses on the construction of *ideal* sized data chunks. A Space Filling Curve based data placement strategy is used to

ensure near-maximum data concurrency at the second level. Smart-IO has been implemented and evaluated as part of the ADaptive I/O System (ADIOS) framework [1] developed by Oak Ridge National Laboratory (ORNL). We use ADIOS to leverage its penetration amongst existing scientific applications and to exploit the ADIOS file format (BP) as the format for Smart-IO data. Our experimental evaluation was conducted on the Jaguar Cray XT5 [2] supercomputer at ORNL to demonstrate that Smart-IO is able to provide both good balance and high performance for challenging access patterns in scientific applications. In our test case for S3D we obtain a 72x speedup to the planar read compared to the logically contiguous data layout, while introducing negligible overhead in data generation.

II. Background and Related Work

A. Background

A scientific simulation typical consists of three major components, namely computation, I/O (checkpoint-restart) and data post-processing. One combination of computation and I/O is called a *time step*. As shown in Figure 2, a simulation runtime normally includes multiple timesteps. After the simulation finishes, data post-processing, such as data analytics and visualization, is performed intensively on the simulation outputs for scientific discoveries. As a major portion of application turnaround time, I/O performance plays a significant role in determining simulation productivity and energy efficiency. This study focuses on data post-processing and seeks for strategy to enable fast read.

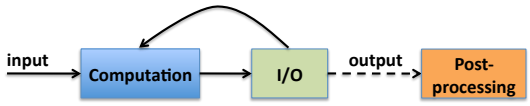


Fig. 2: Major Components of Scientific Computing

In order to improve the read performance, a thorough understanding of application access patterns is crucial. Figure 3 shows a visualization result from a S3D [6] combustion simulation output. Such output is normally organized as multidimensional variable, as shown in Figure 4(a). While the checkpoint-restart usually reads entire variables, the majority of the data analytics are performed on a subset of the data, such as an arbitrary orthogonal **full plane** (Figure 4(c)), or an arbitrary orthogonal **subvolume** (Figure 4(b)) More complex reading patterns can be described through a composition of these three patterns, or through minor variations.

Among these patterns, significant performance variations are observed for the query on a subset of data, particularly on an orthogonal plane. Such phenomenon is often referred as *dimension dependency* [25], where the read performance depends on the dimensions of the query, rather than the data size of the query.

B. 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. Read performance has gained more attention lately. [19] and [14] evaluated and discussed the performance of many of the reading patterns for extreme scale science applications. A number of studies [12], [7], [28], [27] 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. To speed up read performance, many multidimensional array declustering algorithms [18], [20], [4], [5] were proposed to improve common access patterns of a multidimensional array. Schlosser et al. [23] explored the chunk placement strategy at the disk level. More efficient access was observed by putting data chunks on the adjacent disk blocks. Deshpande et al. [9] and Fan et al. [10] examined how read performance can be further improved by chunking and proper caching. However, the future access pattern for scientific application varies and may not be known *a priori*. A data layout should accommodate any access pattern. Chunking can be further combined with partitioning, a.k.a *subchunking*, which further decomposes the data chunk. In [21], Sarawagi and Stonebraker gave an initial guidance of what is the proper way for chunking. Then Sawires et al. [22] and Sorouch et al. [24] proposed different multilevel chunking strategy to further improve the performance for range queries on a multidimensional array. Otoo et al. [17] 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.

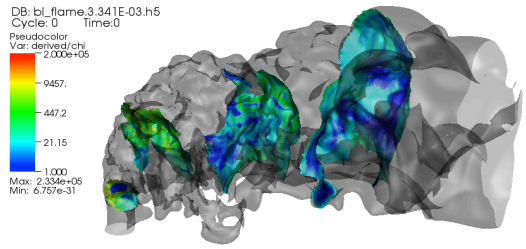
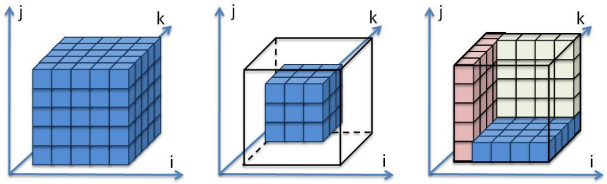


Fig. 3: A S3D Combustion Simulation Result



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

Fig. 4: Scientific Data Organization and Common Access Patters (k: fastest dimension)

III. Optimized Chunking Model and Smart-IO Approach

Smart-IO uses a combination of four techniques to speed up scientific data analytics: **Optimized Chunking** model, **Hierarchical Spatial Aggregation** (HSA), **Dynamic Subchunking** (DYS) and **Space Filling Curve** (SFC) reordering.

Figure 5 shows software architecture of Smart-IO and its components.

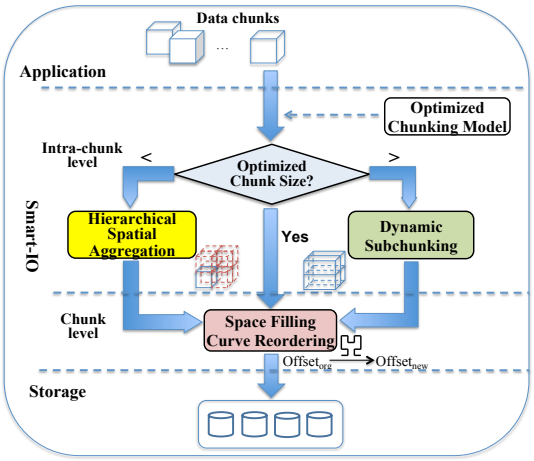


Fig. 5: Two-Level Data Organization of Smart-IO

At the high level, Smart-IO sits between the application layer and the storage system. It provides a two-level data organization. The first level is the intra-chunk level. This level focuses on building the data chunks into *Optimized Size*, which is derived from our *Optimized Chunking* model. It strikes for a good balance between the data transfer efficiency and processing overhead through specific system parameters. For data chunks that are not satisfying the Optimized Size, corresponding reconstruction is performed by using Hierarchical Spatial Aggregation or Dynamic Subchunking. 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 ensure the close-to-maximum data concurrency from the storage system. Under such organization, a data chunk has three paths moving towards the storage system. The rest of this section describes the design of these components in detail.

A. Optimized Chunking Model

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. Optimized Chunking model is to theoretically investigate the *ideal* chunk size for a multidimensional data on a HPC system.

Intuitively there is a *sweet spot* for the chunk size where the overhead on the seek operation (occurring when we attempt to read too many small chunks) and redundant data retrieval (occurring when we attempt to read a few large chunks) achieves the best balance for the slow dimension. Such a sweet spot is where the optimal read performance can be expected. Significantly varying the chunk size towards either direction results in performance degradation as the I/O becomes burdened by overhead or dominated by disk seeks. Given a 3-D variable as shown in Figure 4(a), we show the performance for planar read over the chunk size in Figure 6, where N_{ocs} is the sweet spot.

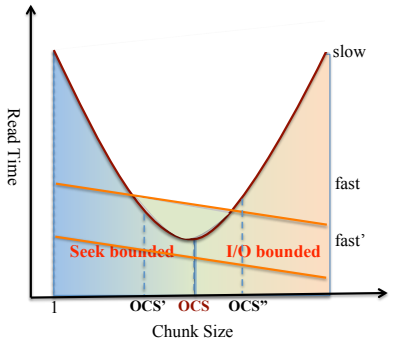


Fig. 6: The Read Time vs. the Chunk Size

Because dividing large chunks breaks the contiguity on the fast dimension, the read performance degrades proportionally with the decreasing chunk size. This results in two scenarios as shown in Figure 6, represented by two orange straight lines. In view of the general performance on all the dimensions, the fastest total read time may not incur at the point *OCS* but still within the *Optimized Region* of *OCS'* and *OCS"*. The performance difference inside such region is within a small margin. As this study is aimed at finding an *optimized* chunk size, we use our solution of *OCS* as the guidance for data organization. Our experimental results in Section IV demonstrate that this value provides satisfactory performance.

After a series mathematical derivation (the detail of the derivation can be found in [26], we have the Optimized Chunks Size:

$$OCS = BW_{io} \times (CC + T_s) \times \alpha \quad (1)$$

where $BW_{i/o}$ is the I/O bandwidth, T_s is the time unit for each seek operation, and CC is the communication cost unit. α represents the interference factor on the large-scale system. However, in order to determine this factor, a thorough study of the system is needed, which is not the focus of this work. For a simplified analytical model, the external and internal interferences to the storage system are ignored. Such modeling can help pinpoint a solution that enables near-optimal I/O performance tuning in a timely fashion.

B. Hierarchical Spatial Aggregation

Even though 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 is 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 a Hierarchical Spatial Aggregation strategy 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 to be aggregated, the aggregator process will be selected in a round-robin fashion within the same group for load balancing. Figure 7 shows an example of aggregating one variable from 16 processes in a 2-D space. For every spatially adjacent 4 processes, an *AG* is formed, where process 0 is selected as the first aggregator. 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 aggregators who have hold all the data of its group members in their memories. In our case, process 0 is chosen as the second level aggregator. After aggregation, only the aggregators will be writing out the data. Figure 8 gives an example of data movement and file output for 3 variables where 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 read requests and seek operations are reduced by $level \times 2^n$ times, where level is the level of HSA performed.

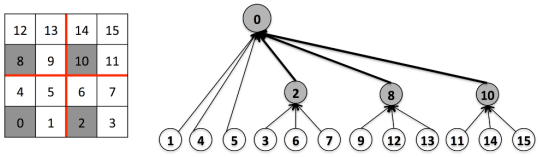


Fig. 7: Hierarchical Spatial Aggregation

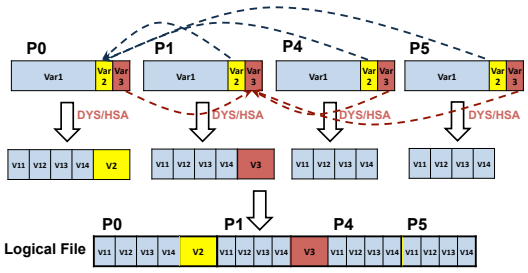


Fig. 8: Data Movement and File Output

C. Dynamic Subchunking

To optimize the read performance for large data chunk, we design Dynamic Subchunking to divide each large data chunk into *subchunks* of the optimized size. Subchunking the large data chunk is important for applications that access data with a high degree of locality.

However, how to decompose a chunk needs to be investigated. Assume a 2D chunk is divided into 9 subchunks, there are three common options for such decomposition. Figure 9(b) to Figure 9(d) provide examples of these options. The red arrow represents seek operation. The shaded region represents the amount of data needs to be read in for a request on slow dimensions. The row major is the fast dimension and column major is the slow dimension.

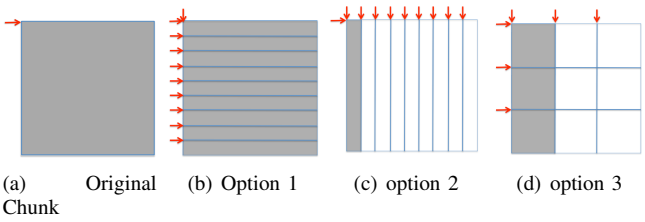


Fig. 9: Dynamic Subchunking

Comparing Figure 9(b) to Figure 9(d), subchunking on the slow dimension does not benefit reading on that dimension but introduce 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. Moreover, reading on the fast dimension is typically more efficient because data is laid out contiguously. For example, reading 120MB data on Jaguar is expected to cost less than half second with one process. Assume 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 second. This is still within the tolerable margin, comparing to more than 60 seconds for read time on the slow dimension as measured. Therefore, *subchunking is performed on all the dimensions except the slowest dimension for an n-dimensional data chunk. The number of subchunks on each dimension is balanced as much as possible.*

D. Data Organization based on Space Filling Curve

When the Optimized Chunks are constructed, a Hilbert Space Filling Curve [11] ordering is used to rearrange the placement of data chunks on storage. Such design is to address the data concurrency issues when a subset of data is to be retrieved from only a small number of storage targets with traditional linear placement strategy. For example, Figure 10 shows that the read requests in the column major is limited to the bandwidth of only one storage node by linear placement. To ensure that close-to-optimal concurrency is extracted from the storage system for a variety of data access scenarios, we apply a Hilbert curve-based placement strategy which shuffles the placement of each data chunk along the Hilbert curve ordering. As we can see in Figure 10, data concurrency is improved three times on the slow dimension with Hilbert curve reordering compared to the original linear placement strategy.

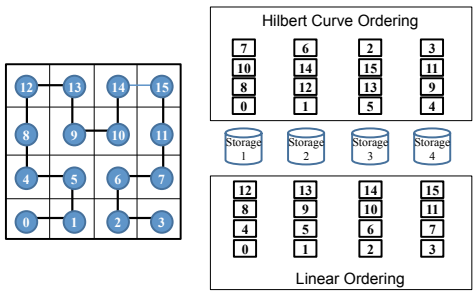


Fig. 10: Comparison of Linear Placement and Hilbert Curve Reordering of 16 Chunks on 4 Storage Nodes

IV. 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 [3], [13], [15], [28], [19] for optimized I/O performance. By default ADIOS applys chunking for multidimensional arrays. We evaluated Smart-IO on the *Jaguar* supercomputers , currently the third fastest supercomputer in the world [16] located at ORNL. Jaguar is equipped with Spider (an installation of Lustre) for the storage subsystem, In our experiments, we used the Widow 2 partition of Spider which contains 336 storage targets (OSTs).

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. We set up the output file to contain 4 variables (Var1, Var2, Var3 and Var4) with dinstict sizes. Table I shows the data chunk size after the original 3-D domain decomposition Based on the system parameters and Equation (1), we calculated the Optimized Chunk Size as 2.5MB on Jaguar. The correspondent Smart-IO data reorganizations performed on each chunk 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
Operations	DYS/SFC	DYS/SFC	SFC	HSA/SFC

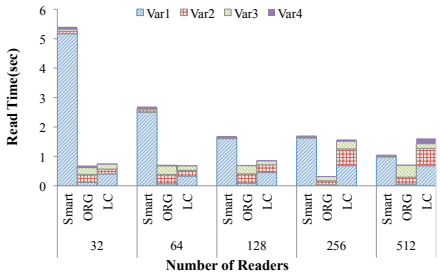
The performance evaluation of Smart-IO is mainly focused on the I/O performance of planar read, which is the most common yet very challenging access pattern. We measure the read performance among three types of data organization strategies: Logically Contiguous (LC), the chunking strategy of the original ADIOS (ORG), and two-level data organization of Smart-IO (Smart). A separate test program is created to evaluate the performance of logically contiguous data layout.

We evaluate the performance of reading one plane from all the variables within the test file, a common scenario in data analytics and visualization. The original data was written by 4,096 processes. The total read time are shown in Figrure 11.

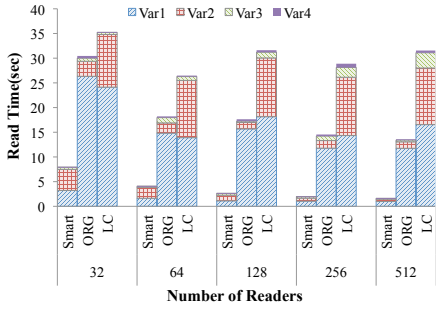
As we can see, LC shows advantage only when data is contiguous and smaller amount of readers are used. Due to limited data concurrency on fast dimension, the performance of LC degrades with more readers, as shown in Figure 11(a). As observed in Figure 9, Smart-IO showed small performance degradation compared to the original ADIOS on fast dimension due to subchunking on Var1 and Var2. However, by using an OCS-based two-level data organization, Smart-IO is able to outperform the original ADIOS significantly on the slow dimensions. Overall, a maximum of 8 times and 43 times speedup of total read time is achieved compared to the original ADIOS and LC, respectively.

V. Contribution

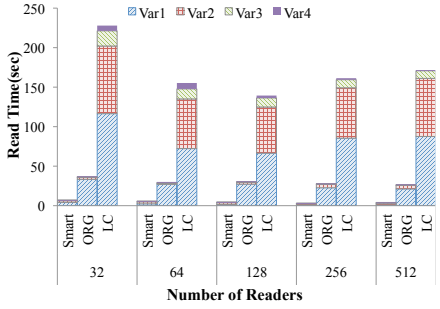
Our work addresses the read performance issue of large-scale scientific applications. It focuses on providing a system-



(a) jk



(b) ik



(c) ij

Fig. 11: Planar Read Performance of Multiple 3-D Variables (k is the fastest dimension)

aware data organization that can effectively utilize the underlying storage system. We have designed and implemented a light-weighted I/O scheme 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. By optimizing the units for data chunks and utilizing the Hilbert curve placement strategy, a much more balanced and consistently good read performance is ensured for scientific post-processing. Our experimental results on Jaguar Supercomputer at ORNL demonstrate 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.

Smart-IO also begins to show real-world impact in scientific computing field. As a part of ADIOS I/O framework, Smart-IO is scheduled in the next release of ADIOS in the summer of 2012, so that scientific applications can benefit from the *smart* data organization bought by Smart-IO.

REFERENCES

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

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

[3] 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.

[4] 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.

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

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

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

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

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

[14] 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.

[15] 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.

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

[17] 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.

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

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] Y. Tian and W. Yu. Finding the optimized chunking for multidimensional array on large-scale systems. Technical Report AU-CSSE-PASL/12-TR01, Auburn University, 2012.

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

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