2017 IEEE International Parallel and Distributed Processing Symposium Workshops

An Application-Aware Data Replacement Policy for Interactive Large-Scale Scientific Visualization

Lina Yu, Hongfeng Yu
University of Nebraska-Lincoln
Lincoln, Nebraska, USA
Email: {lina, yu}@cse.unl.edu

Hong Jiang
University of Texas at Arlington
Arlington, Texas, USA
Email: hong.jiang@uta.edu

Jun Wang
University of Iowa
Iowa City, IA, USA
Email: jun-wang-1@uiowa.edu

***Abstract*—The unprecedented amounts of data generated from large scientific simulations impose a grand challenge in data analytics, and I/O simply becomes a major performance bottleneck. To address this challenge, we present an application-aware I/O optimization technique in support of interactive large-scale scientific visualization. We partition a scientific data into blocks, and carefully place data blocks in a memory hierarchy according to a characterization of data access patterns of user visualization operations. We conduct an empirical study to explore the parameter space to derive optimal solutions. We use real-world large-scale simulation datasets to demonstrate the effectiveness of our approach.**

***Keywords*-I/O optimization; scientific visualization; data replacement; large-scale data.**

I. Introduction

Researchers have devoted substantial efforts to develop efficient executions of simulations on petascale or next-generation exascale machines. However, it remains a grand or even greater challenge to visualize and interpret large simulation data. First, I/O speed is significantly lagging behind computing speed and the data size generated from scientific applications keeps increasing [11]. Transferring large data across a deep memory hierarchy to carry out visualization calculations simply becomes the main bottleneck for an interactive visualization pipeline. Second, data access patterns can be dramatically different from user visualization operations and be dynamically changed across hierarchical memory levels. Therefore, it is particularly difficult to identify data access patterns for large-scale interactive visualization.

Many sophisticated techniques have been developed to address the fundamental data locality issue and improve data movement efficiency. However, most traditional approaches target generic solutions, less consider application characteristics, and thus are difficult to achieve optimal performance for large-scale scientific applications. Recent efforts have been perceived to incorporate domain knowledge to optimize data movement and enhance the scalability of end-to-end scientific workflows [18], [15]. However, these approaches mainly investigated data access patterns of simulations, and only considered relatively simple data analysis operations.

In this paper, we first characterize data access patterns of visualization, and leverage application knowledge to derive a novel scheme to predict data access during user interactive operations. Our prediction method is effective even when a user randomly or nearly randomly accesses data in 3D. Based on the prediction results, we develop a data replacement policy to exploit data locality and minimize data movement across multiple levels of a memory hierarchy.

We have evaluated our approach on machines with multiple hierarchical memory levels and compared it with the traditional methods, including First-In-First-Out (FIFO) and Least Recently Used (LRU). The experimental results have shown that our method can achieve superior performance in support of interactive visualization.

II. Related Work

The out-of-core techniques have been exploited in I/O-efficient volume rendering, isosurface computation, and streamline computation [12]. For example, in order to optimize streamline tracing in large unstructured data, Ueng et al. [16] proposed a top-down out-of-core preprocessing algorithm that built an octree partition to restructure unstructured grids and then loaded the octree cells on demand. However, this method required to replicate a cell on the octree nodes that the cell may intersect. Leutenegger and Ma [7] proposed to use R-trees to solve the imbalance problem in the structure of octree to optimize searching operations on large unstructured datasets. Pascucci and Frank [10] used a space-filling curve for data layout and indexing that can be simply and efficiently computed by bit masking, shifting and addition. This method can be easily used for the multi-resolution computation of arbitrary slices of very large datasets. Sutton and Hansen [14] proposed the T-BON (Temporal Branch-On-Need Octree) technique for fast extraction of isosurfaces of time-varying datasets. Isenburg et al. [6] used mesh simplification as an example to show how to adapt out-of-core mesh processing techniques to perform their computations based on the new processing sequence paradigm. Although these out-of-core techniques make it possible to visualize a large dataset with a reduced I/O cost, most of them require expensive pre-processing

978-1-5386-3408-0/17 $31.00 © 2017 IEEE
DOI 10.1109/IPDPSW.2017.16

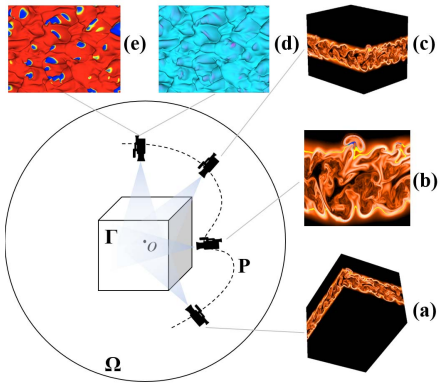


Figure 1. A scientific visualization process using a combustion simulation data.

steps, such as conducting some form of data duplication, or building multi-resolution representations.

Caching and prefetching heuristics are also commonly used to reduce the number of the cache miss and the I/O cost.They have been extensively investigated for computer systems. The main aim of these studies is to minimize different cost metrics, such as miss ratio, total cost, and average latency. Belady [1] discussed replacement algorithms based on a virtual-storage concept for automatic memory allocation. Hutchinson et al. [5] demonstrated a duality between prefetching and queued writing with parallel disks, involving read-once accesses and interleaved access to striped sequences. Dash and Demsky [2] presented a distributed transactional memory system that attempted to automatically hide network latency by speculatively prefetching and caching objects. Megiddo and Modha [9] proposed a cache replacement policy named Adaptive Replacement Cache that maintained two LRU lists of pages to increase hit ratio. Although these techniques can significantly improve I/O performance, there are few caching and prefetching techniques targeting highly interactive visualization applications.

III. Background

A. Interactive Scientific Visualization

The data generated from a scientific simulation is typically volumetric, multivariate, and time-varying. Volume visualization techniques are often used to depict different aspects of variables and their possible relationships. Figures 1, 2 and 3 illustrate two interactive volume visualization processes using the datasets of a combustion simulation and a climate simulation as examples. We can see that visualization operations can be roughly categorized into two types.

The first type of operations is *view-dependent*. As shown in Figure 1, given a volume data Γ in $\mathbf{R}^3$, a user can explore it in a spherical domain $\Omega \subset \mathbf{R}^3$ enclosing Γ. We assume that Ω and Γ are concentric, and the center is o. The user can move a camera along a path $\mathbf{P}$ inside Ω,

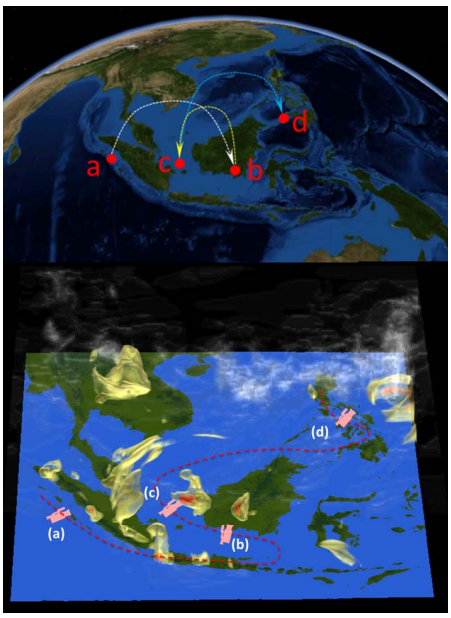


Figure 2. A scientific visualization process using a climate simulation data. The dotted line in the top image denotes an overview of a sample camera path around the earth. The bottom image provides a top view of the camera path in the climate simulation data, where the interplay between smoke (colored in yellow to red) and typhoon (colored in white) is visualized.

and use rendering techniques, such as volume rendering [8] to generate an image of Γ at a camera position on $\mathbf{P}$. For example, Figure 1 (a) and (c) show the variable of stoichiometric mixture fraction (*mixfrac*) from two camera positions along $\mathbf{P}$. Figure 1 (b) shows a zoom-in image corresponding to a camera position closer to this data.

The second type of operations is *data-dependent*. Apart from moving the camera position, a user can also apply visualization techniques (e.g., *transfer functions* [4], query-based visualization [3], etc.) to control the visual properties (e.g., visibility, color, shape, etc.) of different data variables and regions. For example, Figure 1 (d) and (e) correspond to a top view of the data. The image (d) shows an iso-surface of *mixfrac* colored by the variable of scalar dissipation and the image (e) shows the same iso-surface but colored by the variable of OH radical.

Moreover, after perceiving certain regions of interest, scientists often conduct detailed data analysis and visualization using the combination of numerous queries based on possibly complex functions of the primary variables. Take climate simulations for instance in Figure 2, the bottom image shows a mesoscale modeling of smoke (colored in yellow to red) transport over the southeast Asian maritime continent and the interplay of typhoon (colored in white). Scientists can explore the dataset along a camera path as shown in the red dotted line. An overview of the camera path can be easily perceived from the earth in the upper image.

The images (a)-(d) in Figure 3 illustrate the views from

four different view angles and view positions on this path. Scientists may want to collect statistical information useful to describe the relationship between variables. As shown in Figure 3, the statistic analysis results can be dynamically generated and displayed with each image. They are three histograms showing the distribution of water vapor mixing ratio (QVPOR), wind magnitude, and a correlation matrix of 151 primary variables for the regions seen from the images. These results can be used to assist scientists in their analysis tasks [13], and gain interactive feedbacks according to their views.

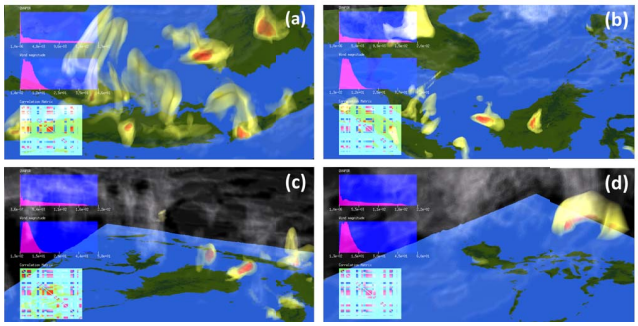


Figure 3. The images (a)-(d) are the four views generated from the different view angles and positions along the path. At the left of each image, there are two histograms and a correlation matrix corresponding to the distribution of QVPOR, wind magnitude and the correlation among 151 primary variables for the regions seen from the image. These analytic graphs are dynamically updated according to views.

B. I/O Challenges

With ever increasing computer power, simulations produce increasingly large quantities of data to be visualized, which has imposed severe challenges to carry out interactive visualization with these *view-dependent* and/or *data-dependent* operations. A typical issue is that a volume data generated from a large-scale simulation cannot be entirely loaded into the memory layer (e.g., main memory or GPU memory) immediately accessed by a processing unit (e.g., CPU or GPU). A commonly used strategy is to only load the visible data regions that could be considerably smaller than the entire data. Furthermore, when a user moves the camera position, the visible data regions are also changed. Thus, it requires to quickly move data between faster (but smaller) memory and slower (but larger) memory, while maintaining an interactive rendering speed. However, such data movement is a major bottleneck in today's many large-scale scientific applications. Real-time data visualization and analysis based on different view positions and view directions require an efficient data transferring and placement solution.

Computer graphics and visualization communities have developed sophisticated *view-dependent* algorithms to optimize data movement according to a user's view [12]. The basic idea is to first build a multi-resolution representation of

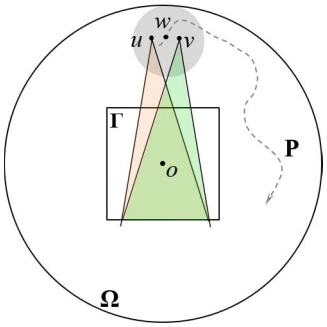


Figure 4. The orange (green) region corresponds to the view frustum or the visible region of the camera at the position u (v).

data, and then for a data region is far from the camera, only its coarser representation needs to be loaded and rendered. This idea is based on a simple fact that the viewed area is reduced for an object moving away from the camera. Conventional data replacement policies, such as LRU, are used to keep the data region most recently seen in faster memory.

However, for *data-dependent* operations, the full extent of visible data is often required to generate an accurate depiction of data (e.g., the shape and color of the iso-surface, or the statistics of a variable within a region of interest) for gaining detailed scientific insights. These operations are based on the functions of variables, and this fact implies that we must consider every data element to accurately evaluate the functions, such as the coloring of the iso-surface in the images (e) and (d) of Figure 1 and the calculation of the histograms and correlation matrix shown in the images (a)-(d) of Figure 3. However, these complex functions are typically a priori unknown and not easily invertible. Furthermore, there is not enough space in the memory for loading the whole data of all variables to compute the correlation matrix. This presents a difficulty for realizing performance either by indexing the variables in a preprocessing step, or by traditional multi-resolution approaches that may defeat the original purpose of performing high-resolution simulations. In addition, data access patterns in visualization are also highly influenced by camera paths that can be randomly or nearly randomly formed by a user exploration. Thus, it is very challenging to predict data access patterns using conventional data-dependent solutions.

IV. Our Approach

We present a novel approach to predict data access patterns and minimize data movement by bridging *view-dependent* and *data-dependent* strategies. Compared to existing techniques, our approach can significantly improve the effectiveness of data *caching* (i.e., reusing data blocks in faster memory) and *prefetching* (i.e., overlapping data movement with computation). Our approach is based on two observations:

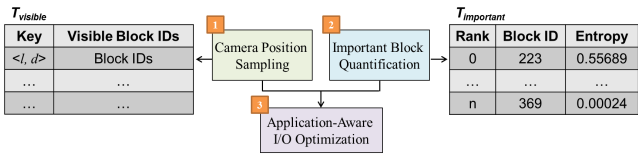


Figure 5. The overview of our method.

Observation 1: The visible data regions can be very similar among the camera positions within a certain small vicinal area in Ω. As shown in Figure 4[1], the view frustums (i.e., the visible regions) of two nearby camera positions, u and v, are largely overlapped. This implies that, although a camera path **P** can be rather random within Ω, the view frustum of any position w on **P** can be approximated by a nearby point u or v in Ω, where u or v may not be exactly on **P**. This property can be used to improve the effectiveness of data caching, which is to maximize the reuse of data blocks in faster memory. As there may exist overlapping data, we could prefetch the data that currently is not on the fast memories for next view point on a camera path.

Observation 2: Scientists often use their domain knowledge to narrow their exploration of certain regions of interest. A considerable amount of data can be pre-filtered out for their study. For example, for the climate simulation data shown in Figure 3, scientists mainly focus on the regions that are severely contaminated by PM10 (particulate matter colored in yellow to red) and are interacted with the typhoon (colored in white), and may neglect the ambient regions. This implies that we can derive a measure to quantify the importance of data, and arrange the placement of data along a memory or storage hierarchy according to data importance. Specifically, we may not need to load the data of an ambient region frequently, and can store them in a slower memory while we can place an important region in a faster memory closer to the processing unit.

A. Overview

Figure 5 shows the overall process of our method that has three steps. Without loss of generality, we assume that a volume data is divided into a set of uniform-size blocks.

In Step 1 (Section IV-B), based on Observation 1, we sample camera positions in Ω according to view directions and distances with respect to the center of Ω. We then construct a look-up table, $T_{visible}$, where the key of the table, $< l, d >$, is a tuple of view direction l and distance d of a sampling camera position and each key corresponds to the visible data blocks.

In Step 2 (Section IV-C), based on Observation 2, we quantify the importance of each data block using an entropy measure, and pre-load the important blocks to relatively faster memory levels. A table, $T_{important}$, containing the importance information of data blocks will be constructed

[1]For clarity, we use the 2D square and circle in the figures and examples.

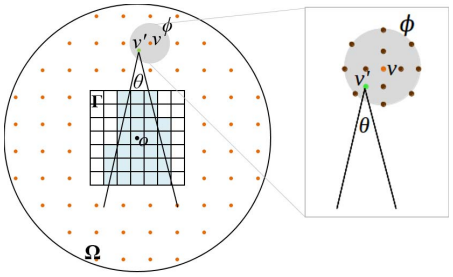


Figure 6. Camera position sampling. Each orange point represents a sampling position v. ϕ is a small spherical domain centered at v. For a green point v' inside ϕ, the light blue blocks intersect with the visual frustum of the camera at v', and thus are visible at v'. The view angle of the frustum is θ.

for data prefetching process. Specifically, after building $T_{visible}$ based on the sampling camera positions, we only prefetch visible data blocks with high importance instead of the entire set of blocks if the total size of predicted data blocks exceeds the cache size in fast memory. These two steps are performed as one-time pre-processing before visualization.

In Step 3 (Section IV-D), $T_{visible}$ and $T_{important}$ are used to optimize I/O during rendering with possibly dynamically changed transfer functions and view positions. In our I/O optimization, for each position on a camera path, we look up $T_{visible}$ to load the corresponding visible blocks to faster memory for rendering. We also leverage $T_{important}$ to pre-load the important blocks to faster memory. Thus, during rendering, only a few blocks will be replaced by LRU policy. Our method can better predict future data requests and make a more informed prefetching decision to reduce I/O cost and support highly interactive visualization.

B. Camera Position Sampling

Based on Observation 1, we develop a method to predict the data blocks that are visible from a certain position on a camera path.

We first sample a number of camera positions in Ω, as shown in Figure 6. In general, the larger number of camera positions in Ω we sample, the higher accuracy the prediction will have. However, for a larger number of sampling positions, the total time of building the table $T_{visible}$ will be longer, and the look-up time for prefetching data in Step 3 will be longer as well, thus increasing the overhead.

To determine an appropriate number of sampling camera positions, we conducted a test based on a random camera path with the view direction changes within 10-15 degrees. We fixed the other parameters, and only changed the number of sampling positions in Step 1. Figure 7 illustrates the test of the comparison of miss rate and I/O time (i.e., the time to load the missed data block) between different numbers of sampling positions on four datasets shown in Table I. Each camera position v corresponds to a view direction $l = \vec{vo}$ and a distance $d = \| \vec{vo} \|$, where o is the centroid of Ω.

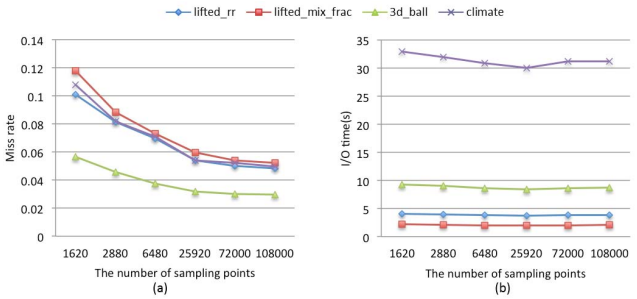


Figure 7. (a) and (b) show the comparison of miss rate and I/O time between different number of camera position sampling, respectively.

As seen from Figure 7 (a), the higher number of sampling points, the lower miss rate we can obtain. However, as shown in Figure 7 (b), the I/O time with $25,920$ sampling points has the lowest I/O time, and the I/O times with $72,000$ and $108,000$ sampling points are higher. This is because the saving from a reduced miss rate can not suppress the increase of query time in a larger look-up table. Thus, the I/O overhead will be higher.

We then compute the visible blocks within a vicinal area of each sampling camera position v. We build a small spherical domain ϕ centered at v as shown in Figure 6. We sample several points v' inside ϕ. The number of sampling points v' in a vicinal area mainly depends on the radius of this area. The zoom-in part in Figure 6 shows the sampling points in ϕ. Given a view angle θ of the view frustum of the camera, we assume the angle between a block b and a camera position v' is φ. If φ is less than $\theta/2$, then b intersects with the frustum and thus is visible. We can easily compute φ as:

$$\varphi = arccos(\frac{\vec{v'b_i} \cdot \vec{v'o}}{\| \vec{v'b_i} \| \| \vec{v'o} \|}), \quad (1)$$

where b_i ($i \in [0,7]$) are the coordinates of eight corner points of b, and o is the centroid of the volume, $\vec{v'b_i}$ and $\vec{v'o}$ are two vectors, and $\|\cdot\|$ is the L-2 norm operator. After computing all visible blocks for each point v' within ϕ, we use a union operation to gather the set S_v of unique visible blocks for v. The key, which is a tuple of the view direction $l = \vec{vo}$ and a distance $d = \| \vec{vo} \|$ combined with its corresponding visible data block set S_v, will be inserted into the look-up table $T_{visible}$. By that analog, we construct $T_{visible}$ after we compute the set S_v of visible blocks for each position v. Each entry in $T_{visible}$ is a key-value pair, where the key is the tuple $< l,d >$ denoting a sampling position v, and the value is S_v. This table is only computed once as a pre-processing step. Moreover, it is independent to specific datasets and only depends on the views and the total block numbers of a volume.

The selection of the radius r of ϕ determines the accuracy of our prediction of data blocks during visualization. As shown in Figure 6, we aggregate the view frustums of v' and compute the visible blocks accordingly. If r is too large, the aggregated view frustum of ϕ can cover the whole volume, and thus incur over-prediction. Similarly, if r is too small, we under-predict the visible blocks.

In addition, the radius r of ϕ must be larger than the distance between two camera positions, because our goal is to predict the blocks that will be used for the next camera position on the camera path. That is, the vicinal area of each sampling position should contain the next camera position on a camera path. According to our observation, an ideal case is that the total size of the predicted and current visible blocks is equal to the cache size in faster memory, so that we can take full advantage of faster memory and more accurately prefetch the visible blocks for the next view point on a camera path. As shown in Figure 8, we hope that the blue blocks, corresponding to the predicted visible region from the ϕ of a camera position v, can be held in faster memory. We can estimate the size of blue blocks by computing the aggregated frustum (the green region) in Figure 8. Therefore the radius r of ϕ is decided by the view angle θ of the frustum, the view direction l, the view distance d, and the cache size of faster memory. Among these parameters, the view distance d can be changed more dynamically during interactive visualization. Intuitively, if the camera is far from the volume, the visible region is larger than the case when the camera is near from the volume. Thus, the radius r should be adjusted according to the view distance d to improve the accuracy of our prediction of data blocks during rendering.

The distance between two view points on a camera path is also an important factor impacting on the radius r of ϕ, because we hope the vicinal area ϕ of each sampling camera position could contain the next camera position on a camera path. To satisfy this condition, we may need a bigger radius r, which however will contribute to over-prediction and cause the size of predicted data blocks larger than fast memory. To address this problem, we can fully make use of the importance information of data blocks which will be introduced in detail in Section IV-C. We only select the most important blocks in the set S_v for each sampling point rather than keeping all the blocks in S_v in the look-up table $T_{visible}$. Section V-B2 will detail the parameter choices of r.

The data block size also plays an important role in the performance of our algorithm. We need to consider the trade-off between the number of I/O operations and the size of accessed data blocks. We experimented different block sizes to minimize the I/O overhead. The detailed discussion of choices of block size will be provided in Section V-B1.

C. Important Block Quantification

Locating the most commonly used blocks at faster memory is the key for fast visualizing large data, as timely results are critical to many applications such as on-demand query and real-time visualization. For the initial placement of data,

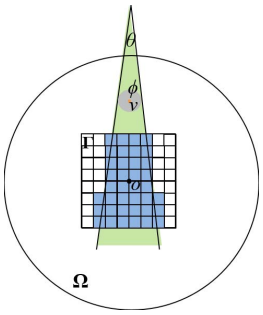


Figure 8. The model to select a radius of a small spherical domain ϕ.

instead of placing all data blocks on slower memory, we estimate the important blocks and pre-load them to faster memory.

In this work, we select Shannon's entropy calculation in information theory to quantify the importance of a block. Shannon's entropy $H(x)$ defines the information content of a random variable x as:

$$H(x) = -\sum_{x \in X} p(x) log p(x), \quad (2)$$

where $p(x)$ is the probability mass function of x. This is a measure of the uncertainty about a given random variable and also indicates how much information the dataset contains [17]. This provides a measure to identify important data blocks of the volume, and has a large contribution towards the total uncertainty of a group of blocks, because the blocks with a small variation in their values often denote ambient regions, which usually have similar intensity values. The blocks with high entropy have high likelihoods of presenting important areas in the volume dataset. For example, in a combustion simulation dataset, the regions of which values have greatest changes tends to be the most interesting part for domain scientists. We build the table $T_{important}$ by sorting the values of entropy for all the blocks, which allows us to select important blocks and put them on faster memory.

As discussed in Section IV-B, the importance information for each data block can be used to predict visible data blocks when the next view point is relatively far from the previous one on a camera path and when a large vicinal area around a sample point would result in over-prediction. In this case, we should only insert the block IDs with higher entropies into the look-up table $T_{visible}$ for the prediction of visible regions in Step 1, so that we can identify the blocks with a higher possibility to be used for the next view point. This method facilities interactive visualization when users conduct *data-dependent* operations (e.g, tuning transfer functions) by their domain knowledge.

D. Application-Aware I/O Optimization

Based on our camera position sampling and important block quantification, we develop an application-aware I/O optimization technique for large scientific visualization, which can achieve a lower miss rate and a higher I/O speed than traditional methods. This algorithm consists of three major phases as shown in Algorithm 1.

Algorithm 1 Application-aware I/O optimization in large-scale scientific visualization

```
1:  Let vec_b record visible block IDs from a view point
2:  Let vec_fast record block IDs on fast memory
3:  Let num_block record the number of blocks
4:  Let time[num_block] record the latest used time of each block
5:  Initial vec_b ← φ, time[num_block] ← -1
6:  Load T_visible and T_important
7:  Load the block IDs whose entropy values greater than a threshold σ in T_important into vec_fast
8:  for each view point v_i on a camera path P do
9:    for each block b_j in the volume do
10:     if b_j is visible then
11:       vec_b ← b_j
12:     end if
13:   end for
14:   for each block b_j in vec_b do
15:     if b_j is not in vec_fast then
16:       Fetch b_j from slow memory and replace a block in vec_fast with the lowest value in time and its value in time should be less than i
17:     end if
18:     time[b_j] = i
19:   end for
20:   // Overlap rendering and prefetching
21:   Render the visible blocks
22:   During rendering, find the nearest sampling view point v'_i from v_i in Ω; look up T_visible to find the blocks corresponding to the key v'_i, and prefetch the ones with the entropy values greater than a threshold σ; during prefetching, replace the blocks in vec_fast that have the lowest values in time and their values in time are less than i
23: end for
```

First, we conduct initialization and pre-load important blocks to fast memory according to our important block quantification (Lines 1-7).

Second, during a user interactive exploration, for each view point v_i on a camera path **P**, we compute the visible blocks using our camera sampling. Then we check if all visible blocks are stored on the fast memory close to the computing unit. If not, we fetch the blocks from the slow memory and the least recently used items are replaced (Lines 8-19).

Third, after all blocks are located in the fast memory, the rendering operation starts. If we use traditional methods such as FIFO and LRU to conduct data placement, I/O is idle during the rendering time because there is no

Table I
DATASETS USED IN OUR EXPERIMENTAL STUDY.

name	description	resolution	#variables	size
3d_ball	a synthetic dataset	$1024 \times 1024 \times 1024$	1	4GB
lifted_mix_frac	a combustion simulation dataset	$800 \times 686 \times 215$	1	472MB
lifted_rr	a combustion simulation dataset	$800 \times 800 \times 400$	1	1GB
climate	a climate simulation dataset	$294 \times 258 \times 98$	244	7.2GB

prediction scheme to prefetch the data for the next step. In our algorithm, we take full advantage of rendering time to prefetch the data which may be used to the next view point on the camera path. Given a random camera path **P**, for each view point v_i on **P**, we find the nearest sampling point v_i' from v_i in Ω, so that it is easy to prefetch the most frequent blocks according to the look-up table $T_{visible}$ before rendering, and replace the blocks which are least recently used on the current view point. Although prefetching all the blocks to the fast memory could dramatically improve the prediction accuracy and decrease the miss rate for the next rendering pass, it might lead to a high overhead. To further enhance the performance, we can fully overlap prefetching and rendering, and combine the importance information in $T_{important}$ with our prefetch scheme, which is to prefetch the blocks with the entropy values of greater than a specified threshold σ (Line 21-22). Thus, the total processing time will be significantly lower than traditional methods.

V. RESULTS AND DISCUSSION

We present the experimental results of our I/O optimization method on the performance of volume rendering using a synthetic dataset and several real simulation datasets of different resolutions. We evaluated our method with different camera paths, transfer functions, and sampling camera positions. The conventional FIFO and LRU methods were used in our comparison study.

A. Experimental Datasets and Environment

Table I lists the datasets used in our experiments. A synthetic dataset *3d_ball* models a 3D ball with continuous changes of intensity inside. *lifted_mix_frac* and *lifted_rr* are two real combustion data sets, and *climate* is a time-varying climate data set. All datasets consist of 4-byte floating-point values. Two different types of camera path were used in our experiments. One type of camera path is a spherical path with different degree intervals for camera positions, while another type of camera path is a random path with different degree changes for each camera position. The total number of sampling positions along a camera path is 400.

In our experimental study, we used a desktop with an 8X Intel Core i7 3.6GHz CPU. We used a three-level memory hierarchy containing 16GB DRAM, a 512GB solid-state drive (SSD) and a 3TB hard disk drive (HDD). We tested the I/O performance across these three levels. We employed a GPU-accelerated volume rendering, and applied our data

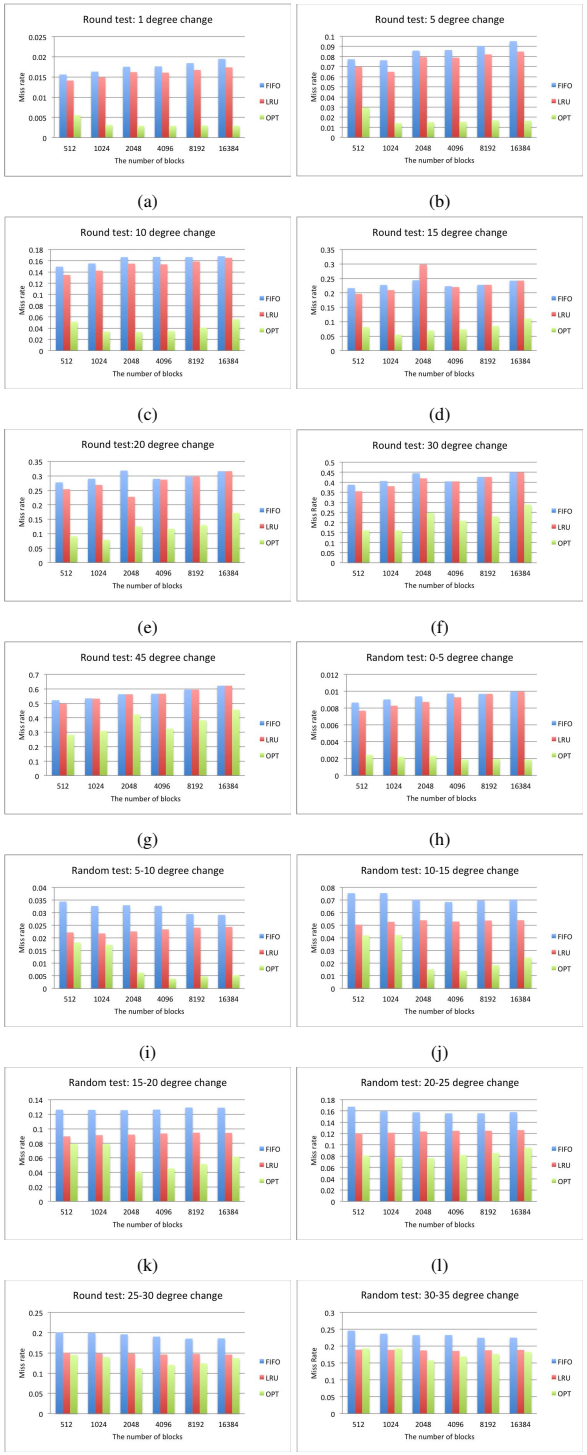


Figure 9. Miss rate between different block divisions.

replacement for transferring a dataset from the HDD to the SSD to the DDRM. We neglected the data transferring between the DRAM and the GPU memory in this study. The

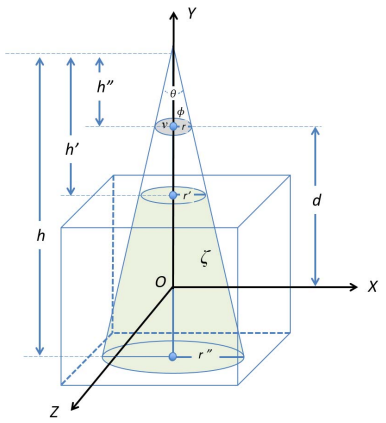


Figure 10. A mathematical model for deciding the radius of spherical area around each sampling points.

ratio of cache size is 0.5 between two successive memory levels (i.e., for a dataset stored on the HDD, the cache sizes on the SSD and the DRAM are 50% and 25% of the dataset size, respectively).

Our goal is to lower the I/O time by reducing the miss rate across the levels of the memory hierarchy. Although the summation of the prefetching time and I/O time might be larger than the traditional method, the prefetching time can be overlapped and hidden by rendering time. Thus, miss rate is a very important metric in our experiments. For all experiments, we measured the total miss rate and the I/O time across DRAM, SSD and HDD, and the total time combined with the I/O and the rendering. We compared our method with the FIFO and LRU methods. Our method refers to as OPT in the figures.

B. Parameter Choices

There are two parameters which most influence the performance of our optimized algorithm: the block size and the radius of spherical area around each sampling point when we build a camera position sampling look-up table. We conducted experiments to study the influence of these parameters on the performance of our algorithm.

1) Block size: The block size influences the time to load data blocks into memory. To study the impact of this parameter, we performed a sequence of experiments using the *3d_ball* dataset. The block size tested were $32\times32\times64$, $32\times64\times64$, $64\times64\times64$, $64\times64\times128$, $64\times128\times128$, $128\times128\times128$. We compared the miss rate between our optimized method and the FIFO and LRU methods. Figures 9 (a)-(g) illustrate the miss rate tests based on a spherical camera path with the view direction changes of 1, 5, 10, 15, 20, 25, 30 and 45 degree per sampling camera position, respectively. Figures 9 (h)-(n) demonstrate the miss rate tests based on a random camera path with the view direction changes between 0-5, 5-10, 10-15, 15-20, 20-25, 25-30 and 30-35 degree per sampling camera position, respectively.

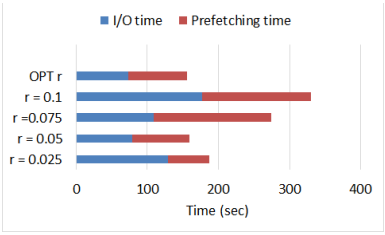


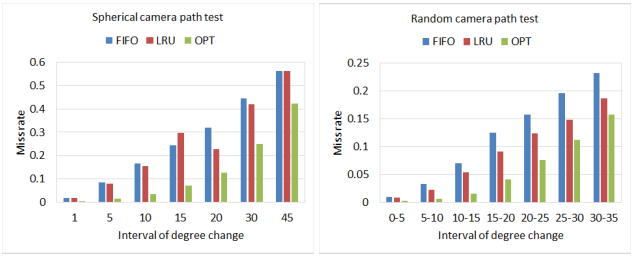
Figure 11. The total I/O and prefetching time over 400 camera positions using the optimal r computed by our method and the pre-defined r values.

Figure 9 clearly shows that our method is significantly superior to FIFO and LRU no matter how many blocks are divided. When the camera position changes within a small range such as 0-5 degree on either a spherical or random camera path, we can see that smaller block sizes reduce the total miss rate. However, when the camera view direction changes within a larger range, different block sizes do not show a considerable difference. This is because the miss rate mainly depends on the data replacement across a memory hierarchy and the smaller block size will increase the total number of replaced blocks. We found that the range of the total block number between 1024 to 4096 can reduce the miss rate. The determination of block size is also a trade-off between the number of I/O operations and the size of data read. We recommend that selected block sizes are multiples of the read buffer size.

2) Radius of spherical area around a sampling point: The radius r of spherical area ϕ around each sampling point (Section IV-B) is another important parameter in our method. In order to choose a suitable value for this parameter, we build a mathematical model to estimate a good range, as shown in Figure 10. We define a Cartesian coordinate system with the origin at the center of the volume. The edge size of the volume is normalized to 2, and the corresponding coordinates are from -1 to 1. Assume v is a sampling point with a distance to the origin is d. For each small sampling point in a small spherical region ϕ with the radius r, we can construct a frustum between two parallel planes of the volume. If we aggregate the frustums of all the sampling points in ϕ, we can obtain a bigger frustum ζ (the light green region in Figure 10) with a view angle θ and the two radius lies between the volume are r' and r'', respectively. As we hope fully take advantage of fast memory, the ideal situation is to put the whole frustum between the volume in fast memory. In other words, the volume of the frustum ζ should be less than or equal to the size of fast memory. This give us:

$$\frac{\pi r''^2\frac{h}{3}-\pi r'^2\frac{h'}{3}}{8}=\frac{cache\ size\ of\ fast\ memory}{cache\ size\ of\ slow\ memory}, \tag{3}$$

where $\pi r''^2\frac{h}{3}-\pi r'^2\frac{h'}{3}$ is the volume of ζ, and 8 denotes the normalized volume size. According to Figure 10, we can



(a) (b)

Figure 12. Miss rate across a spherical path (a) and a random path (b).

easily derive:

$$r' = \frac{rh'}{h''} = tan(\frac{\theta}{2})h' \\ = tan(\frac{\theta}{2})(d-1+\frac{r}{tan(\frac{\theta}{2})}) \quad (4)$$

and

$$r'' = tan(\frac{\theta}{2})h = tan(\frac{\theta}{2})(d+1+\frac{r}{tan(\frac{\theta}{2})}) \quad (5)$$

Thus, we can replace r' and r'' in Equation 3 and obtain the optimal r as:

$$r = \sqrt{\frac{4 \times cache\ size\ of\ fast\ memory}{\pi \times cache\ size\ of\ slow\ memory} - \frac{1}{3}tan(\frac{\theta}{2})^2} - d \times tan(\frac{\theta}{2}) \quad (6)$$

We can see that if we fix the view angle and the ratio of cache size between fast memory and slow memory, the optimal r needs to be dynamically computed according to the distance d.

To verify the correctness of Equation 6, we performed a test using the $lifted\_rr$ dataset that was partitioned into 1024 blocks with the block size $50 \times 100 \times 50$. We fixed the view angle and applied a camera path with 400 positions. The normalized volume edge size is 2. We compared the I/O time and the prefetching time between the optimal r computed by Equation 6 and the pre-defined r values 0.1, 0.075, 0.05, and 0.025 with respect to the normalized volume edge size. As shown in Figure 11, our method archived the lowest amount of time for I/O and prefetching. In practice, a user may often zoom-in or zoom-out during an interactive visualization, resulting in a dynamically changed d value. In this case, our method can automatically compute the optimal r value tailored to different d values.

C. Effect of Camera Paths

Figure 12 (a) shows the comparison of miss rate across a spherical path with different degree intervals using the $3d\_ball$ among three methods: FIFO, LRU, and our optimized method. The $3d\_ball$ dataset is divided into 2048 blocks. As shown in Figure 12 (a), a spherical path with 1-degree change per camera position has the least miss rate among three methods and the miss rate using our optimized method is the one-fourth of the miss rates of the other two methods. With a larger degree change per camera position, the miss rate gradually increases. The reason is that a smaller view direction changes, the less the number of replaced blocks is. The miss rates of our method are less than half of the miss rates of FIFO and LRU.

Figure 12 (b) compares the miss rates among FIFO, LRU, and our method from different degree changes per camera position on a random camera path on the $3d\_ball$ dataset, which is divided into 2048 blocks. The path has 400 camera positions with randomly different d and l values. The miss rates were calculated over these directions. The result shows that our method has lower miss rates, almost one third miss rate of FIFO and half of LRU.

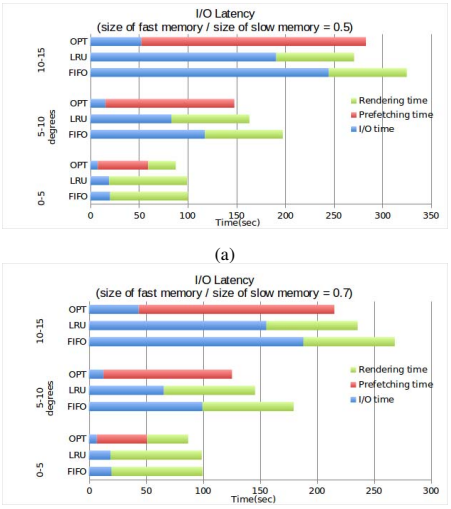


Figure 13. The total I/O and prefetching time with different ratios of cache size over 400 camera positions using FIFO, LRU, and our optimized method.

D. I/O Latency

To show the I/O latency, we tested the total time including the I/O, prefetching and rendering times on $3d\_ball$ dataset with 4096 blocks. Figure 13 compares the results of three methods with different view direction changes on a random camera path. In our method, the total time is equal to the summation of the I/O time, and the maximum of the prefetching time and the rendering time. This is because the prefetching time using our optimized method can be overlapped by the rendering time for the most cases. Using FIFO or LRU, the total time is equal to the summation of the I/O time and the rendering time.

In Figure 13 (a), the ratio between the fast memory size and the slow memory size is 0.5. When the view direction changes within 10 degrees, our total time can be decreased up to 12% than the LRU method and 25% than the FIFO method. However, our total time appears longer when the view direction change are larger than 10 degrees. This is because the replaced blocks increase with respect to larger

view direction changes. Given a limited fast memory size, more predicted visible blocks may not be held in the cache. In this case, the prefetching time is longer for loading missed predicted visible blocks.

To tackle this situation, we can enlarge the cache size. Figure 13 (b) shows the latency results when we set up the ratio between the fast memory size and the slow memory size is 0.7. In this setting, more predicted visible blocks can be held in the cache. Our method achieves the lowest total time even the view direction changes are within 10-15 degrees, and the speedup is 8.6% to LRU method and 19.7% to FIFO method.

VI. Conclusions and Future Work

In this work, we characterize data access patterns in interactive large-scale scientific visualization. The unique features of our solution are to model the similarity of data access in a 3D visualization space and to leverage the data importance derived from domain knowledge. Therefore, our solution can achieve an optimal I/O time even for a user's random exploration. In our experimental study, we tested different parameter choices and verified the effectiveness of our designed models. The performance of our method is superior to the conventional methods used in visualization.

In the future, we would like to extend our method for parallel data fetching and rendering. In particular, we plan to study data partitioning and distribution schemes by leveraging data importance information. We also will experiment our method on visualization with different user interaction techniques, such as virtual reality with head-mounted displays. These use cases may require a faster interactive response, and impose more challenging I/O stresses on runtime large-scale data processing.

VII. Acknowledgment

This research has been sponsored by the National Science Foundation through grants IIS-1423487, ICER-1541043, and CNS-1116606. J. Wang is supported by the NASA Interdisciplinary Science Program (grant award #: NNX16A50G) managed by Hal B. Maring. The combustion datasets are provided by Dr. Jacqueline Chen at Sandia National Laboratories.

References

[1] L. A. Belady. A study of replacement algorithms for a virtual-storage computer. *IBM Systems journal*, 5(2):78–101, 1966.

[2] A. Dash and B. Demsky. Integrating caching and prefetching mechanisms in a distributed transactional memory. *Parallel and Distributed Systems, IEEE Transactions on*, 22(8):1284–1298, 2011.

[3] M. Glatter, J. Huang, S. Ahern, J. Daniel, and A. Lu. Visualizing temporal patterns in large multivariate data using modified globbing. *Visualization and Computer Graphics, IEEE Transactions on*, 14(6):1467–1474, Nov 2008.

[4] C. D. Hansen and C. R. Johnson. *Visualization handbook*. Academic Press, 2011.

[5] D. A. Hutchinson, P. Sanders, and J. S. Vitter. Duality between prefetching and queued writing with parallel disks. In *European Symposium on Algorithms*, pages 62–73. Springer, 2001.

[6] M. Isenburg, P. Lindstrom, S. Gumhold, and J. Snoeyink. Large mesh simplification using processing sequences. In *Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, page 61, 2003.

[7] S. Leutenegger and K.-L. Ma. Fast retrieval of disk-resident unstructured volume data for visualization. *External Memory Algorithms and Visualization*, 1998.

[8] M. Levoy. Display of surfaces from volume data. *IEEE Comput. Graph. Appl.*, 8(3):29–37, May 1988.

[9] N. Megiddo and D. S. Modha. ARC: A self-tuning, low overhead replacement cache. In *FAST*, volume 3, pages 115–130, 2003.

[10] V. Pascucci and R. J. Frank. Global static indexing for real-time exploration of very large regular grids. In *Supercomputing, ACM/IEEE 2001 Conference*, pages 45–45. IEEE, 2001.

[11] J. Shalf, S. Dosanjh, and J. Morrison. Exascale computing technology challenges. In *Proceedings of the 9th international conference on High performance computing for computational science*, VECPAR'10, pages 1–25, 2011.

[12] C. Silva, Y.-J. Chiang, J. El-Sana, and P. Lindstrom. Out-of-core algorithms for scientific visualization and computer graphics. *IEEE Visualization Course Notes*, 2002.

[13] J. Sukharev, C. Wang, K.-L. Ma, and A. T. Wittenberg. Correlation study of time-varying multivariate climate data sets. In *2009 IEEE Pacific Visualization Symposium*, pages 161–168. IEEE, 2009.

[14] P. M. Sutton and C. D. Hansen. Accelerated isosurface extraction in time-varying fields. *Visualization and Computer Graphics, IEEE Transactions on*, 6(2):98–107, 2000.

[15] D. Tiwari, S. Boboila, S. S. Vazhkudai, Y. Kim, X. Ma, P. Desnoyers, and Y. Solihin. Active flash: towards energy-efficient, in-situ data analytics on extreme-scale machines. In *FAST*, pages 119–132, 2013.

[16] S.-K. Ueng, C. Sikorski, and K.-L. Ma. Out-of-core streamline visualization on large unstructured meshes. *Visualization and Computer Graphics, IEEE Transactions on*, 3(4):370–380, 1997.

[17] C. Wang and H.-W. Shen. Information theory in scientific visualization. *Entropy*, 13(1):254–273, 2011.

[18] F. Zheng, H. Yu, C. Hantas, M. Wolf, G. Eisenhauer, K. Schwan, H. Abbasi, and S. Klasky. Goldrush: resource efficient in situ scientific data analytics using fine-grained interference aware execution. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 78. ACM, 2013.