A Hierarchical Tensor-Based Approach to Compressing, Updating and Querying Geospatial Data

Linwang Yuan, Zhaoyuan Yu, Wen Luo, Yong Hu, Linyao Feng, and A-Xing Zhu

Abstract—With the rapid development of data observation and model simulation in geoscience, spatial-temporal data have become increasingly multidimensional, massive and are consistently being updated. As a result, the integrated maintenance of these data is becoming a challenge. This paper presents a blocked hierarchical tensor representation within the split-and-merge paradigm for the compressed storage, continuously updating and data querying of multidimensional geospatial field data. The original multidimensional geospatial field data are split into small blocks according to their spatial-temporal references. These blocks are represented and compressed hierarchically, and then combined into a single hierarchical tree as the representation of original data. With a buffered binary tree data structure and corresponding optimized operation algorithms, the original multidimensional geospatial field data can be continuously compressed, appended, and queried. Data from the 20th Century Reanalysis Monthly Mean Composites are used to evaluate the performance of this approach. Compared to traditional methods, the new approach is shown to retain the quality of the original data with much lower storage costs and faster computational performance. The result suggests that the blocked hierarchical tensor representation provides an effective structure for integrated storage, presentation and computation of multidimensional geospatial field data.

Index Terms—Multidimensional data modelling, data compression, data updating, blocked hierarchical tensor representation

1 INTRODUCTION

TEOSPATIAL field data, such as remote sensing images **J** and large climate model simulation data, are becoming multidimensional and massive [1]. These data have large volumes (e.g. several gigabyte) and high dimensionality (e.g. dozens or hundreds of attributes). Large amounts of observation of exiting attributes/variables are uninterruptedly generated by global observation systems [2]. These data are often compressed for storage. The newly arrived data should also be continuously compressed and appended to the existing data in such a way that the newly added should be integrated to the existing data and made as a whole with the existing data. In addition, this updating process should be completed in a short time and can be repeatedly applied for the next patch of new data. The compression and storage should maintain the consistency of the spatial-temporal reference (STR) of this data. Balance among the data accuracy, data compression performance and convenience for indexing, querying and analysis is required.

The explosion of both the data volumes and dimensionality of these geospatial field data makes the storage, management, query and processing a daunting challenge to existing solutions [3]. Multidimensional data are stored and accessed linearly in memory and on hard disk but should be randomly queried and updated from any dimension. Classical methods use data indexes (e.g., Pyramid, R-Tree, VA-Files) to accelerate the query and storage [4]. These data indexes split data into segments and then map the segments to the linear ordered data I/O sequence [5]. When the dimension grows, both the data segmentation and the data structure are becoming complex and inefficient. Big data or data-intensive computing solutions use parallel data I/O and computation to accelerate the data accessing and updating [6]. However, large number of computers and complicated computation architectures are required to provide the I/O bandwidth and computation power needed [7].

The situation becomes worse when the continuously data compressing, appending and updating are required. Within the existing data representation and analysis framework, neither the classical methods nor the big data or data-intensive computing solutions are suited for dynamic data appending and updating [8]. And finding alternative data structures which fit the underlying storage architecture and are easy for data appending, compression and querying at the same time might be challenging [9]. To our best knowledge, nearly all exiting solutions for continuously data processing require different data structures during the management, query and analysis procedures and need to undergone several complex processing steps before they reach the stage as final, manageable, searchable and computable data

L. Yuan is the with the Key Laboratory of Virtual Geographic Environment, Ministry of Education and Jiangsu Provincial Key Laboratory of NSLSCS, Nanjing Normal University, No. 1 Wenyuan Road, Nanjing, China. E-mail: yuanlinwang@njnu.edu.cn.

Z. Yu, W. Luo, Y. Hu, L. Feng, and A.X. Zhu are with the Key Laboratory of Virtual Geographic Environment, Ministry of Education, Nanjing Normal University, No. 1 Wenyuan Road, Nanjing, China. E-mail: {yuzhaoyuan, huyong, axing}@njnu.edu.cn, luow1987@163. com, 414228780@qq.com.

Manuscript received 12 Dec. 2013; revised 4 June 2014; accepted 6 June 2014. Date of publication 12 June 2014; date of current version 23 Dec. 2014. Recommended for acceptance by C-Y. Chan.

For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TKDE.2014.2330829

^{1041-4347 © 2014} IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

products [10]. The frequent data transfer between different data structures in addition slows down the processing throughput.

Tensor is an indispensable tool for multidimensional data processing and analysis [11]. Derived from data-intensive applications, computationally-oriented researchers organize multidimensional arrays as tensor-structured datasets [12], [13]. Tensor decomposition [14], tensor-based PDE solving and signal extraction [13] are then be applied for pattern extraction, high dimensional data manipulations and visualizations [15], [16], [17], [18]. However, most of these tools are for specific computations and have limited functions for data management, query, manipulation and operation [11]. In most solutions, the curse of the dimensionality problem [19] and Null Space problem [20] still exist.

Recently, the hierarchical tensor representations (HTR) (e.g., [21], [22], [23], [24], [25]) provide completely new tensor approximation of high dimensional data expression and rich reference for high dimensional data analysis. Hierarchical tensor representation has the property of coordinates-independence, and it has the inherited linkage between tensor algebra, calculus and partial differential equations [26]. Thus, hierarchical tensors representation provides us the ideal mathematical structure and analysis tools to manage and analyze high-dimensional data [27]. Most recently, computational tools, htucker-Toolbox [28], tensor-train Toolbox [21] and Tensor Calculus library [29] are gaining increasing attention [27]. These advances of hierarchical tensorbased computation have already been used in abstract mathematical and engineering numerical computations. Hierarchical tensors, which are proved to be a useful tool for multidimensional computation, also provide the potential to be a powerful tool in multidimensional geospatial field data representation, compression and operation.

However, several critical issues need to be addressed for hierarchical tensors for multidimensional geospatial field data representation and operation. The first is that the dimension of the geospatial field data is mostly determined by splitting of the spatial-temporal references (e.g., latitude, longitude, height and time). The imbalance of the dimension splitting (i.e. the resolution between spatial-temporal dimensions) of the attributes will be a critical obstacle for efficient storage and computation. For example, for a gridded global data, the split of the spatial dimensions ranges from dozens to thousands (e.g., 1440×720 for a $1/4^{\circ}$ grid). Yet the temporal and attribute dimensions vary from two to several thousands. This dimensionality imbalance will rapidly increase the complexity and storage space for typical tensor solutions. Second, geospatial field data are dynamically updated and appended. During the data updating and addition, the spatial-temporal reference should be kept consistently in a standard datum. However, most of exiting hierarchical tensor approaches are optimized for neither dynamic data updating and combining nor maintaining the consistency of spatialtemporal reference. Third, there will be a high risk of encountering the Null Space problem [20]. Due to the imbalance of dimensionality, the dimensions split tensor might be very sparse, and many data units will be null. Fourth, some lowdimensional characteristics will no longer be present in the high dimension. For example, when the dimension is large enough, the nearest distance and furthest distance will not be as significant as they are in the low dimension space. The diminishing difference between the nearest and furthest distances will make the classical data analysis and operations inefficient [30].

The above issues call for a new data structure and new algorithms to support data organization, compressed storage, data appending and query. Near balanced hierarchical representation and organization will be more helpful and efficient than multidimensional tables or arrays for supporting large tensor expression [31]. The dimension imbalance, the sequentially data appending and randomly data access nature of multidimensional geospatial field data require data to be stored in each dimensional structure independently as blocks rather than stored as a whole linearly. In this way, quick and stable data organization, efficient and compressed data storage, continuous data appending can be applied to each block independently.

The integrated use of the split-and-merge paradigm [32] and the hierarchical tensor decomposition [22] provides an ideal solution for multidimensional geospatial field data representation, compression and operation. The split of original geospatial field data in accordance with the spatiotemporal references can produce more balanced multidimensional tensor blocks with spatial-temporal coordinates logged orderly. These tensor blocks can then be represented hierarchically to support both the sequential data appending and the randomly data access. Taking advantage of tensor approximation, tensors of each block can be compressed efficiently. By developing dynamically updatable data representation, efficient hierarchical data structures and data merging/updating algorithms, the continuously addition with compression, updating and querying of multidimensional geospatial field data can be achieved.

In this paper, a hierarchical tensor decomposition based on the split-and-merge paradigm is developed for continuously compression and appending of multidimensional geospatial field data. Our goal is to propose a hierarchical data structure to reformulate and store the large volume of geospatial field data and to develop methods for data storage, query and computation support using this data structure. We illustrate this through a prototype implementation. The prototype has five components: 1) the design of a buffered hierarchical data structure and data decomposition strategies; 2) a proposal for a blocked data separation mechanism for splitting the huge tensors into small blocks according to the spatial-temporal reference; 3) a proposed algorithm that allows for data appending which is free of arithmetical operations and also computationally adaptive with continuous compression; 4) the development of a hierarchical structure-preserving and dimensional-independent data query which needs only to reform the row of the matrix in the leaf node; 5) the provision of computational operators such as tensor addition and linear operations, as well as a hierarchical structure-preserving computational framework.

The preliminaries of multidimensional representation of geospatial field data are given in Section 2. Methods and implementation for blocked hierarchical data representation, combining and data appending, compression and query mechanism are presented in Section 3. The experimental results are presented in Section 4 and conclusions are provided in Section 5.

where

2 PRELIMINARY AND FRAMEWORK

2.1 Tensor Representation of Multidimensional Geospatial Field Data

Geospatial field data are data with geographical coordinates, which can be formulated as attribute cubes associated with their spatial-temporal references.

Definition 2.1 (Spatial-Temporal References). An STR is a single valued one-to-one mapping of the real geographical space into the n-dimensional mathematical space, in the form of $f: G^n \mapsto \mathbb{R}^n$, in which G^n is the coordination space of the real geographical world, and \mathbb{R}^n is the n-dimensional functional space.

The definition of STR here is generalized and compatible with the definition of spatial-temporal dimensions in the spatial-temporal database or STOLAP systems. Thus, it is possible to split the STR with temporal and spatial projection for characteristics analysis. For example, a four-dimensional geographical space can be mapped as $\mathbb{R}^4 = \mathbb{R}^3 \times \mathbb{R}$, where \mathbb{R}^3 is the three-dimensional spatial space composed by latitude, longitude, and height; \mathbb{R} is the temporal coordination of the four-dimensional space. A full subspace representation of a three dimensional tensor $A \in \mathbb{R}^{r_1 \times r_2 \times r_3}$ leads to a linear combination of seven subspaces of $\{\mathbb{R}^{r_1}, \mathbb{R}^{r_2}, \mathbb{R}^{r_3}, \mathbb{R}^{r_1 \times r_2}, \mathbb{R}^{r_2 \times r_3}, \mathbb{R}^{r_1 \times r_2 \times r_3}\}$.

According to the definition of STR, a multidimensional spatial-temporal field with the same STR can be defined by a tuple with multiple field elements, with the form of $\langle ST, F_1, F_2, \ldots \rangle$, where *ST* is the spatial-temporal references of the field, and F_i is the associated attribute domain. Then multidimensional data cube can be directly represented as a tensor as follows:

Definition 2.2 (Tensor Representation of Multidimen-

sional Data). *Given a series of multidimensional array sets,* $SD_i = \{F_{t_1}, F_{t_2}, \dots, F_{t_n}\}$ with equal size, a tensor-based multidimensional data cube can be defined as:

$$SD = \chi \in \mathbb{R}^{n_1 \times n_2 \times \dots \times n_d},\tag{1}$$

where χ is the *d* dimensional tensor and n_1, \ldots, n_d denote numbers of elements with dimension from 1 to *d*, respectively. It is also called the dimensionality of the *i*th mode.

The *n*th element in a sequence is denoted by a superscript in parentheses. For example, the matricization of the tensor χ along the dimension (mode) *n* is denoted by $A^{(n)}$, which is called the *n*th matrix in a sequence. The fiber and slice can be seen respectively as the generalized extension of matrix rows/columns and two-dimensional sections of a tensor in higher orders. According to the tensor definition, a higher dimensional tensor can also be composed of tensors (i.e., the lower dimensional tensors), thus the tensor can be used to define data sets of arbitrary dimensions. For example, we can reorganize the three dimensional tensor (rank-3 tensor) as a big matrix by expanding the tensor through one dimension.

2.2 Hierarchical Tensor Decomposition for Geospatial Data

The subspace combination of original tensor data can be represented as a tree structure, called dimension tree or subspace partition tree [22]. The hierarchical tensor representation according to a specific subspace partition tree [22] is defined as follows:

Definition 2.3 (Hierarchical Tensor Representation). Given a subspace partition tree T_D , a tensor $v \in V = {}_a \otimes_{j \in D} V_j$ with finite index set D. The hierarchical tensor decomposition of v is characterized by finite dimensional subspaces U_D as

$$U_{\alpha} \subset V_{\alpha} := {}_{a} \otimes_{i \in D} V_{j} \text{ for all } \alpha \in T_{D}, \tag{2}$$

$$U_{\alpha} = \begin{cases} U_j, \alpha \in L(T_D) \\ U_{\alpha_1} \otimes U_{\alpha_2}, \alpha \in T_D \backslash L(T_D), \alpha_1, \alpha_2 \in S(\alpha). \end{cases}$$
(3)

With Equations (2) and (3), the hierarchical structure of the entire tensor can be constructed. If bases $B_{\alpha} = [b_1^{(\alpha)}, \ldots, b_{r\alpha}^{(\alpha)}] \in U_{\alpha}$ and $b_{r\alpha}$ are given to generate a subspace representation of U_{α} , it has the following relationship:

$$b_l^{(\alpha)} = \sum_{i=1}^{r_{\alpha_1}} \sum_{j=1}^{r_{\alpha_2}} c_{ij}^{(\alpha,l)} b_i^{\alpha_1} \otimes b_j^{\alpha_2}, \{\alpha_1, \alpha_2\} = S(\alpha),$$
(4)

where $C^{(\alpha,l)} = (c_{ij}^{(\alpha,l)})_{1 \le i \le r_{\alpha_1}, 1 \le j \le r_{\alpha_2}} \in K^{r_{\alpha_1} \times r_{\alpha_2}}$, $1 \le l \le r_{\alpha}$ are the coefficient matrices that define isomorphism. With this definition, we arrive at the hierarchical subspace representation of the original tensor.

2.3 Hierarchical Geospatial Field Data Representation (HGFDR)

With the subspace representation, the STR as well as the attribute data can be represented by combination of subspaces. According to the subspace partition of STR and hierarchical tensor decomposition, we can define the hierarchical geospatial field data representation as following:

Definition 2.4 (Hierarchical Geospatial Field Data Representation). Given a subspace partition tree T_D , a subspace representation of geoscience field tensor data $v \in V =$ $a \otimes_{i \in D} V_j$, the HGFDR of v is defined as

$$U_{\alpha} = {}_{a} \otimes_{j \in D} U_{j} \\ = \underbrace{(U_{1} \otimes U_{2}) \otimes B_{12}}_{U_{12}} \otimes \underbrace{(U_{3} \otimes U_{4}) \otimes B_{34}}_{U_{34}} \\ \otimes \cdots \otimes \underbrace{(U_{n-1} \otimes U_{n}) \otimes B_{(n-1)n}}_{U_{(n-1)n}} \\ = \underbrace{(U_{12} \otimes U_{34}) \otimes B_{1234}}_{U_{1234}} \otimes \underbrace{(U_{56} \otimes U_{78}) \otimes B_{5678}}_{U_{5678}} \otimes \cdots \\ \otimes \underbrace{(U_{(n-3)(n-2)} \otimes U_{(n-1)n}) \otimes B_{(n-3)(n-2)(n-1)n}}_{U_{(n-3)(n-2)(n-1)n}} \\ = \cdots \\ = \underbrace{(U_{12\cdots n/2} \otimes U_{(n/2+1)\cdots n}) \otimes B_{12\cdots n}}_{U_{12\cdots n}},$$
(5)

where $\alpha \in T_D$ is the hierarchical tree node and U_i is the coefficient matrix/tensor of each dimension or mode cluster. B_i is the transfer tensor, which makes the computation balanced and satisfies the relationships as given in Equation (6):

$$U_{\alpha} = (U_{\alpha_1} \otimes U_{\alpha_2}) B_{\alpha}, \alpha_1, \alpha_2 \in S(\alpha).$$
(6)



Fig. 1. The overall framework of hierarchical expression of geospatial field data tensors.

With Equation (5), the matrix expression of the parent node can be recursively reconstructed by the son node and the interior transfer tensor B_{α} . A hierarchical subspace approximation and the recursive reconstruction are used to construct the coefficients of the interior nodes. Since the U_{α} can be solved from Equation (5), only the leaf node matrix U_i and the transfer tensor B_i need to be estimated to construct the hierarchical representation of field data according to Definition 2.4. Thus we have

$$vec(x) = (U_1 \otimes \cdots \otimes U_d) \otimes (B_{12} \otimes \cdots \otimes B_{(d-1)d})$$
$$\otimes (B_{1234} \otimes \cdots \otimes B_{(d-3)(d-2)(d-1)d}) \otimes \cdots \otimes B_{12\cdots d}.$$
(7)

2.4 The Overall Framework

The overall framework of the hierarchical expression and the updating of the geospatial tensor are depicted in Fig. 1. The overall idea of the blocked geospatial representation and compression is to split the original imbalanced geospatial tensor into several more dimensional balanced sub-tensors (blocks). These blocks are represented, compressed and updated individually. Then these separately represented and compressed tensors can be merged to form the hierarchical representation of the original geospatial field data. Then the hierarchical representation of the original geospatial field data can be gueried and computed efficiently. Since the blocks are much smaller and more dimensionally balanced than the original geospatial tensor, the hierarchical representation and compression in the split-and-merge paradigm are not only much more computation and memory efficient, but also more accurate due to better data consistency.

3 THE BLOCKED HIERARCHICAL TENSOR REPRESENTATION

In this section, we designed a split-and-merge paradigm that splits the multidimensional geospatial data into blocks, represents and compresses the blocks with hierarchical tensor decomposition, and then reconstructs the compressed blocks into a single hierarchical representation of original geospatial data. The reconstructed hierarchical tensor representation maintains the subspace structure of tensor and compresses the cost of storage and operations.

3.1 Hierarchical Representation of Attribute Blocks

With the subspace split of STR and the attribution dimension, the multidimensional geospatial field data can be split into blocks. Each block has its own spatial-temporal references. In typical situations, the STR of multidimensional attribution fields is the same and is hidden in the data organization. That is, the multiple attribution data are usually a separate data cube, which can be directly represented by a tensor. With the above assumption, we can split the original tensor into several blocks without disorganizing the order of the STR.

A function SplitToBlocks(T, b), where T is the original tensor and b is the block size, is designed to split the original tensor into several near balanced sub blocks. In SplitToBlocks(T, b), the original tensor data are first split based on the attributes because it is possible that significant differences in data ranges and characteristics between different attributes exist. The split according to the spatial and temporal dimensions is then applied to further reduce the dimension imbalance and make the blocks into an ideal size. Since the dimension of the data varies differently, the users are required to customize the split number for each dimension. To keep the natural structure of the data, the split of the temporal dimension is usually a factor of the data-updating interval. For the spatial dimensions, it is usually a regular block which has the same size in terms of the coordinate space.

Assuming all the blocks of the attribute data in feature space $\mathbb{R}^{A_1 \times A_2 \times \cdots \times A_d}$ are classical tensor data, and attributes a_1, a_2, \ldots, a_d as elements of attribute set A, we can define a binary attribute tree that stores the attribution block as follows:

- **Definition 3.1 (Binary Multidimensional Attribute Tree, BMAT).** Given the order d, a binary attribute tree T is defined by the binary splitting of the attributes from 1, 2, ..., d recursively until each leaf node contains only a single attribute d_i . Thus, any node of the BMAT can be identified by elements of the power set $P\{1, 2, ..., d\}$ that satisfy:
 - 1) The root has full attribute sets of $t_r = \{1, 2, \dots, d\}$.
 - 2) Every leaf node contains only one attribute label, thus $t_l = \{i\}, i \in T$.
 - 3) Every interior node, which is not a leaf, has two sons $t_1 = u = \{i, i + 1, ..., j\}$ and $t_2 = v = \{j + 1, j + 2, ..., k\}$ that form an orderly and almost balanced partition of t, which satisfies $t_1 \cup t_2 = t$ and u < v for all $u \in t_1$ and $v \in t_2$.

The coefficients of the BMAT nodes should be computed very carefully. The determination of the coefficients of the BMAT should make the BMAT meet the following criteria: 1) the BMAT should be consistent with the hierarchical tensor representation; 2) the node of BMAT should be independent and can be easily appended and compressed; 3) after the appending and compression of the nodes of BMAT, the representation of the whole tensor is reconstructed which is also a BMAT. Similar to the approaches in [22], we solve the coefficients of the BMAT by recursively applying SVD for subspace projection.

In BMAT, we split the core tensor into several $n \times m$ matrices, where n and m are numbers of elements for each individual attribute, to form a hierarchical structure. The columns of $U^{(n)}$ can then be produced from the first I_n th columns of the left singular matrix in SVD. The transfer tensor

 B_{α} , which expresses the interaction between two dimensions α_1 and α_2 , can be extracted from the SVD of the (α_1, α_2) matricization. With matrices $U^{(1)}, \ldots, U^{(n-1)}, U^{(n+1)}, \ldots, U^{(N)}$ fixed, the original tensor can be projected into the $\mathbb{R}^{I_1 \times \cdots \times I_{n-1}I_{n+1} \times \cdots \times I_N}$ dimensional space. For this construction, the interior nodes in the hierarchical representation only store the interaction of the direct son nodes. To make the tensor product closed, an additional dimension which reveals the characteristics of the matrix U should be added. Then the interior-node expression is a rank-3 tensor, which expresses the < characteristic > - < leftnode > - < rightnode > interaction.

According Equation (6), we have

$$\left(U_{\alpha_{1}}^{H}\otimes U_{\alpha_{2}}^{H}\right)\otimes U_{\alpha}=\left(U_{\alpha_{1}}^{H}\otimes U_{\alpha_{2}}^{H}\right)\otimes\left(U_{\alpha_{1}}\otimes U_{\alpha_{2}}\right)\otimes B_{\alpha}=B_{\alpha}.$$
(8)

Equation (8) suggests the B_{α} in any level is the product of the subtree in the α level and their two son nodes. Therefore, it produces a recursive construction of the computation of B_{α} .

With Equations (7) and (8), we have the following additional Equations:

$$\begin{pmatrix} U_1^H \otimes U_2^H \otimes \cdots \otimes U_d^H \end{pmatrix} \otimes vec(x)$$

$$= \begin{pmatrix} U_1^H \otimes U_2^H \otimes \cdots \otimes U_d^H \end{pmatrix} \otimes (U_1 \otimes U_2 \otimes \cdots \otimes U_d)$$

$$\otimes (B_{12} \otimes \cdots \otimes B_{(d-1)d}) \otimes \cdots \otimes B_{12 \cdots d}$$

$$= (B_{12} \otimes \cdots \otimes B_{(d-1)d}) \otimes \cdots \otimes B_{12 \cdots d}.$$

$$(9)$$

In Equation (9), all the leaf nodes on the right side have disappeared, and the tensor product of the traverse of the leaf node matrix and the original tensor, that is, a numerical tensor with determined values, is represented on the left side. The resulting tensor can then be represented by the hierarchical tensor in the form of BMAT. The leaf nodes of this BMAT can be further computed with the SVD. By recursively using Equations (7) and (9), we can compute all the coefficients. Since the estimation of each U_i is independent, there is an easy parallelism for large-scale field data representation. If all the interior nodes are expressed by the transfer tensor, the original block tensor can be recursively reconstructed from all the nodes. Any subsets of the original tensor can be partly reconstructed by the subtree. Clearly, this recursive reconstruction is a good property for data indexing.

3.2 The Blocked-HGFDR Algorithm

Two critical termination criteria, accuracy and the data size, are considered in the blocked hierarchical tensor representation. In the blocked hierarchical tensor representation, both the accuracy and the data size are first controlled by the block size b, then the BMAT T and the rank k [22]. The block size b determines the size and dimension of the block tensor, which is the foundation of the selection of BMAT T and the rank k. The BMAT T determines the computation balance. To reduce the complexity, we initialized a list for storing the balanced tree data structure of each block during the decomposition.



Fig. 2. Hierarchical tensor decomposition.

The optimal rank *k* affects the number of coefficients and thus affects the approximation precision and final data size. We developed a rank determination function and used the empirical results (see detailed discussion in section 5.2) for the rank *k* determination. The *CompRank*(T, ε , ϕ) procedure determines the minimal rank *k* according to the approximation error ε and the desired compression ratio ϕ . The optimal rank *k* is the minimal *k* that satisfies the relations:

$$\begin{cases} \varepsilon = \alpha Rank^{-\beta} \\ \phi = \frac{orisize}{aRank^3 + bRank^2 + cRank + d}, \end{cases}$$
(10)

where α , β , a, b, c, d are empirical coefficients that are determined by the complexity and structure of the data. Compared with solutions proposed by [22], we developed the form from a recursive decomposition. The matricization process $Matricize(X, dimrep_i)$ deals with the matricization of X according to the dimension sequence $dimrep_i$ and the dematricization process $demat(U, k_1, \ldots, k_t)$, and represents the reconstruction of the new tensor with a dimension of $[k_1, \ldots, k_t]$ from the matrix U. Algorithm 1 and Fig. 2 illustrated the representation algorithm.

Algorithm 1: Compute the Hierarchical Expression (CompHi(T, *tree*, ε , ϕ , b_k))

Input: Original tensor *T*, BMAT *tree*, Precise ε , Compress_ratio ϕ , Split_Block_Size b_k

Output: The list of hierarchical tensor trees *hT*

//Split the original data according to block size b_k ;

 $T_i =$ SplitToBlock(T, b_k);

 $hT = list(hT[1], hT[2], \dots hT[i]);$

//Transverse each block do hierarchical tensor
representation;

for
$$i \in T_i$$
 do

//Calculate the optimized rank *k*;

```
k = \text{CompRank}(T_i, \varepsilon, \phi);
   //Calculate the leaf nodes;
   Treduce = T;
     hT[i] = InitHt(tree);
   for j \in tree.LeafIDs do
        //SVD of the t-dimensional matricization;
        \widetilde{T}_i^{(t)} = U_t \sum_t V_t;
        hT[i].Node[j] = U_t(:, 1:k);
     C_{j-1} = (U_d^H \otimes \cdots \otimes U_1^H) T_i;
     for j = [tree.Lea fIDs - 1 to 0] do
        for tree.LevelIDs(m)=j do
           if (tree.isLeaf(m))==false then
                \widetilde{M}_{j}^{(m)} = S_m \sum_m V_m;
hT[i].Node[m] = S<sub>j</sub>(:,1:k);
           \widetilde{M}_{j-1} = (\prod_{m \in T_i} B_m^H) C_i;
           B_{root} = vec(M_0);
return hT;
```

In the hierarchical tensor representation of each block, the transfer tensor B_i and the U_i matrices are organized in ascending order according to their associated singular values. A truncation of these coefficients can be applied with a simple selection mechanism similar to the principal components analysis form data compression. With different B_i and U_i , the approximation accuracy is determined by the hierarchical rank $(k_t)_{t \in T}$, and the hierarchical ranks have singular values that fulfill the boundaries $\sqrt{\sum_{i>k_t}\sigma_i^2} \le \varepsilon/\sqrt{2d-3}$ [22]. Therefore, an optimal truncation must exist that fulfills the $||A - A_H|| \le \varepsilon$. With the help of Equation (9), the SVD truncations are applied recursively to compute the decomposition. If the selected characteristics are large enough, the data can be approximated for any given accuracy. For real compression, the coefficient selection can be applied according to the data storage capacity and the expression accuracy.

3.3 Combining Blocks to Hierarchical Representation of Original Tensor

The result of algorithm 1 is the hierarchical representation of a series of blocks, which should be combined to retrieve the representation of the original tensor. Since we hope the combined hierarchical representation of tensors is also a hierarchical representation, it is an ideal solution to keep the BMAT structure and calculate the BMAT coefficients of the original tensor directly from the hierarchical representation of the blocks.

To achieve the block combination, we first define the tensor appending in the original tensor form. Given two tensors $A \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_i^{n_1} \times \cdots I_n}$ and $B \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_i^{n_2} \times \cdots \times I_n}$, the *i*-dimension-data appending is defined as $append(A, B) \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_i^{n_1+n_2} \times \cdots \times I_n}$.

A direct initiation of updating tensor data simply combines the tensors through the dimension. In the hierarchical representation of the blocks, the row and column of each U_i contain the length of dimension and the feature size at each dimension, respectively. The data appending from dimension *i* increases the length of dimension *i*, but without changing the length of other dimensions. Since the data elements are accumulated in all dimensions, the feature data are extended. We thus have the following rules for data appending:

- 1) For the data appending dimension, the new U_i is performed by column expanding as $U_i = \begin{pmatrix} U_{iA} & 0 \\ 0 & U_{iB} \end{pmatrix}$;
- 2) for the other dimensions, only the row combining $U_i = [U_{iA} \ U_{iB}]$ is performed;
- 3) for all interior nodes, the data interaction should be extended to make the tensor product closed. Therefore, $B_i = \begin{pmatrix} B_{iA} & 0 \\ 0 & B_{iB} \end{pmatrix}$. With these three rules, a data-appending algorithm can be constructed by decomposing the geospatial field data into hierarchical representation and combining each node. For example, the dimensional data appending from the second dimension can be defined as

$$[AB]_{i} = \begin{bmatrix} U_{1A}U_{1B} \end{bmatrix} \otimes \begin{bmatrix} U_{2A} & 0 \\ 0 & U_{2B} \end{bmatrix} \otimes \begin{bmatrix} B_{12A} & 0 \\ 0 & B_{12B} \end{bmatrix}$$
$$\otimes \begin{bmatrix} U_{3A}U_{3B} \end{bmatrix} \otimes \begin{bmatrix} U_{4A}U_{4B} \end{bmatrix} \qquad (11)$$
$$\otimes \begin{bmatrix} B_{34A} & 0 \\ 0 & B_{34B} \end{bmatrix} \otimes \begin{bmatrix} B_{1234A} & 0 \\ 0 & B_{1234B} \end{bmatrix}.$$

A generalized algorithm for the dynamic-data appending is given in Algorithm 2. In Algorithm 2, the procedures Comb(x, y) and DiagComb(x, y) perform the direct column combination and diagonal combination for the two matrices x and y. According to the hierarchical data structure illustrated in Fig. 2, the combination can be constructed by direct node selection and matrix combination without any arithmetic computation. This property greatly improves the efficiency of data appending. Since the data can be appended from any dimension with the same style, the hierarchical data representation provides a convenient structure for complex data appending.

Algorithm 2: Dynamical Data Block Combining & Updating(Combine(hT, *tree*, ε , ϕ))

Input: list of hierarchical tensors hT, Inserting							
tensor <i>InT</i> , BMAT tree <i>tree</i> , Precise ε ,							
Compress_ratio ϕ							
Output: The combined tensor <i>hNewT</i>							
$hNewT = hT_0;$							
for $i = 1$ to <i>hT.count</i> do							
for $j = tree.nodecount-1$ to 0 do							
$OrNd = hNewT.Node[j]; InNd = hT_i.Node[j];$							
if (tree.isLeaf(j)) then							
if (OrNd.size()== InNd.size()) then							
hNewT.Node[j]=Comb(OrNd, InNd);							
else							
hNewT.Node[j]=DiagComb(OrNd, InNd);							
else							
hNewT.Node[j]=DiagComb(OrNd, InNd);							
Return <i>hNewT</i> ;							



(a) Hierarchical tensor tree construction

Fig. 3. The memory data structure and its updating flow.

4 **CONTINUOUS DATA COMPRESSION, UPDATING** AND QUERY

4.1 The Buffered Binary Tree Data Structure

The major issue in the situation of continuously data appending and updating is the coordination of the compression/updating of the appending block and the compression/updating of the exiting entire data. The compression/updating of the entire data are often much slower than that of the block data. A unified data structure, the buffered binary tree, is developed to achieve the coordination of the compression/updating of the newly appended data and the entire data. The buffered binary tree was constructed by adding buffer spaces to each node of the binary tree. Thus, operations such as inserting, updating and queries operate data directly through the corresponding buffer area at each level rather than transverse all the way from the root to the leaves [33]. Since the buffers require much less space compared with the whole hierarchical tree and are much more frequently written, they could be stored in high speed storage such as memory or the Solid State Disks, while the huge binary tree, which gets updated much less frequently and occupies a massive amount of space, could be placed on the hard disks. Therefore it is not only write-optimized but also memory efficient.

According the combination process of the BMAT (Algorithm 2), the tensor data appending from one dimension with the other dimension kept consistent can be performed by node-to-node BMAT combining. Fig. 3 illustrates how this buffered binary tree can be used in the continuously data appending and updating. The initial binary tree stores the original entire data represented as BMAT. The newly arrived data in the form of BMAT are appended using Algorithm 2. This combination will be very quick but memory consuming because no mathematical computation is performed. When a certain criterion of the memory usage of the appended data is achieved, the data in the binary tree are compressed. Meanwhile, the newly appended data are also continuously appended to the buffer with another thread. When the compression of the data in the binary tree is finished, the BMAT stored in the buffer are then continuously appended to the tree nodes again. With the buffer, the time cost of the hierarchical tensor representation of blocks and the compression of the entire data can be balanced.

4.2 Continuous Data Updating with Compression

The hierarchical subspace representation of tensor provides a possible solution for the compression of massive geospatial field data. With only leaf-node coefficients stored, the overall subspace hierarchy and the original data can be reconstructed with the tensor product. Since the tree structure is independent of the number of elements with different dimensions, the tree structure will be consistent if the dimension number and the relations between different dimensions are determined. This property will lead to an independent updating of coefficients of the tree nodes rather than of the tree structures. Therefore, the BMAT tree structure proposes a linear storage cost and an efficient method for data query, transformation, operation and computation.

The direct data appending requires a large amount of storage space because of the diagonal matrices and tensors constructed. Since the major data redundancy is the co-linearity of the combined U_i matrices and the sparse transfer tensor B_{i} , the compression can easily be constructed by orthogonal projection of both the U_i and B_i to a nested frame, which is achieved by changing the bases. However, the projection of the HGFDR is not direct due to its hierarchical structure. It needs to retain the tensor product meaning and the data accuracy. We therefore first use the Gramian matrix as a bridge to fully support the projections.

A Gramian matrix of a set of vectors v_1, \ldots, v_n in an inner product space is the Hermitian matrix of inner products, whose entries are given by $G_{ij} = \langle v_j, v_i \rangle$ [34]. Assuming $X^t = U_t B_t^H$, we have:

$$X^{t}(X^{t})^{H} = U_{t}B_{t}^{H}B_{t}U_{t}^{H} = U_{t}G_{B_{t}}U_{t}^{H}, \qquad (12)$$

where $G_{B_t} = B_t^H B_t$ is the Gramian matrix of the columns of B. According to Equation (9), we can then develop the projection algorithm for the HGFDR.



Fig. 4. Continues data compression.

Given an orthogonal basis $S_t \in \mathbb{R}^{k_t \times r_t}$ for the r_t dominant eigenvectors of the Gramian matrix G_t , By defining the projection operator $P :\rightarrow Q_t S_t S_t^H Q_t^H$, we have

$$U_t \sum_{t} V^{H}{}_t = U_t = Q_t R_t = U_{Qt} \sum_{Qt} V^{H}{}_{Qt} U_{Rt} \sum_{Rt} V^{H}{}_{Rt}.$$
 (13)

Then, we can define a projection that $P_t P_t^H = Q_t S_t S_t^H Q_t^H$, where S_t contains the r_t dominant eigenvectors of $R_t G_t R_t^H$, Q_t is the Q matrix in the QR decomposition of U_t matrix. So, we have

$$Q_{\alpha}R_{\alpha} = (S_{\alpha 1}{}^{H}R_{\alpha 1} \otimes S_{\alpha 2}{}^{H}R_{\alpha 2}) \otimes B_{\alpha}.$$
(14)

With this equation, we can define the projection computation algorithm (Algorithm 3). In this algorithm, the orthogonalization of the hierarchical tensor is defined with the procedure Mat2orthogonal(hT), and with the help of QR decomposition. The CalGramian(T) procedure computes the Gramian matrix of tensor T [28]. To consistently and continuously apply the computation between different levels, the ModenProd(T, u) is defined. ModenProd(T, u)performs the u - mode matrix product of tensor T and Trans(x) calculates the transposition of matrix x. The overall structure showing how the computation performed with the hierarchical tree structure is illustrated in Fig. 4.

Algorithm 3: Continuously Data Updating and Compression (UpdateComp($hOrT,InTi$, $\triangle t$, ε , ϕ , LMC, B))
Input: Original hierarchical tensor tree <i>hOrT</i> ,
Inserting tensors $InTi$, Updating interval $ riangle$
t, Precise ε , Compress_ratio ϕ , largest allowed
memory cost LMC and buffer size B
Output: compressed hierarchical tensor <i>hNewT</i>
if $CompHi(InTi, tree, \varepsilon, \phi)$. TimeCost $\leq \Delta t$ then
hInTi = CompHi(InTi, tree, ε , ϕ);
else
HorT.buffer [<i>i</i>].append(<i>HInTi</i>);
hInTi = CompHi($InTi$, tree, ε , ϕ);
while $(hOrT.MemCost + InTi.MemCost) \leq LMC$ do
hT = hOrT + hInT;
$hnT = Combine(hT, \varepsilon, \phi);$
hnT = Mat2orthogonal(hnT):



In the buffered binary tree, two independent buffer spaces, which are used to store the U matrix and Gram Matrix during the data updating and appending processes, are constructed for each node independently (Fig. 3). In the initialization stage, the tree nodes and the buffers were all empty. With each SVD truncation, the data value was added to each node of the tree and the initial HGFDR can be constructed. Assuming the data were continually updated with a given time interval $\triangle t$, and each datum was represented in the hierarchical tree, we stored the data in the tree buffer to avoid continuous data I/O. Since the data inserted to the buffer were linear and did not need numerical computation, they were memory- and I/O-efficient. With a given data updating interval $\Delta t_u \leq \Delta t$, the data updating could be done independently of the data-updating interval. This design provides a convenient and simple solution for data updating in memory, which can also be customized according to the memory and CPU conditions. Also, in the continuous compression procedure, the computation of the Gramian matrixes was stored in a separate buffer, and the data in the nodes were first used to compute the Gramian matrixes and then the node values were updated with the Gramian matrixes, in a process similar to the data appending procedure. In this data structure, the query and insert complexity are $O(\log N)$ and $O(\frac{\log N}{B})$, where *B* is the buffer size.

The complexity of Algorithm 3 can be computed according to the following rules:

- 1) The complexity of *d* dimensional original HGFDR of *n* elements and truncation with given rank *k* is $O((d-2)k^3 + k^2 + dnk)$ [28].
- 2) The data appending needs only to combine the data and does not need algebraic computation. In the worst case, a traverse to all the nodes will achieve the full combination; therefore, the worst complexity is $O(\lceil \log_2 d \rceil)$.
- 3) The truncation of the appended data has a complexity of at most $O(dnk^2 + dk^4)$ [28].

To sum the above, the final computational complexity of Algorithm 3 is $O(dk^4 + (d-2)k^3 + (dn+1)k^2 + dnk + \lceil \log_2 d \rceil)$. The complexity is linear with dimension *d* and a fourth-order polynomial with selected rank *k*.

Since the data appending from one dimension is nearly the same as from other dimensions, it is possible to



Fig. 5. Location query and data extraction of HGFDR.

re-arrange the order of the hierarchical-based tensor data expression. It is also possible to insert data slices directly into any location of the original tensor data when the data slices inserted have the same dimension.

4.3 Query for HGFDR

In traditional tensor expressions, the location query should be performed sequentially. In the worst case, it may require that all the data elements be traversed. Fortunately, the data representation in the HGFDR hierarchical tree structure, the recursive reconstruction of the data and the meaningful factor matrix U_i and transfer tensor B_i make each dimension query synchronously possible. Since each row of the U_i contains the order of elements, the location query can be directed by selecting the corresponding row of U_i . A similar rule can be applied to the interior transfer tensor B_i . A graphical expression of the location query of HGFDR is shown in Fig. 5. Since the data in each dimension are independent, the query is independent to the order and can be easily parallelized. Because the search only requires one index for each dimension, the highest possible computation is $O(\sum n_i)$, where n_i is the number of elements in dimension *i*.

Algorithm 4: the range query of HGFDR data based on the subtree (QueryRange(*hT*,*RR*))

```
Input: Hierarchical tensor hT, rectangular range RR

Output: The query tensor T

for i = hT.nLevel-1 to 0 do

for hT.LevelIDs(j)=i do

if (hT.isLeaf(j)) then

hT.Node[j]=submat(hT.Node[j],RR);

else

iLeft = hT.Node[j].lChldID(j);

iRight= hT.Node[j].RChldID(j);

hT.Node[j]=ModenProd(hT.Node[j],hT.

Node[iLeft]);

hT.Node[j]=ModenProd(hT.Node[j],hT.

Node[iRight]);

T = hT.Node[0];

Return T;
```

Another commonly used query of geospatial field data is the rectangular range query for the whole tensor, for example, extracting a certain continuously related region or slice from a certain dimension. The range query is much simpler since it can be performed with a subtree reconstruction. For the range query of data that only affects certain dimensions, the leaf nodes of these dimensions are selected and the interior nodes are reorganized. An HGFDR representation of the subtensor that contains all the data queried can



Fig. 6. The storage data structure of the HGFDR.

therefore be reconstructed. Based on the reconstructed subtensor, the query method can be performed to retrieve the final query result. Algorithm 4 presents the process for the range query of the HGFDR.

4.4 Implementation of In-Memory and Out-of-Memory Storage Structure

Current data storage have not been optimized for multidimensional data access [35]. For efficient storage, data access and operations, a structure should be developed for data storage. Since the primary structure of the HGFDR is a binary tree, the only necessary step to develop the in-memory data structure is to construct an index table to identify the relationship between U_i and B_i , stored as a multidimensional array. Even for the purpose of data updating, the dynamical extension of the U_i and B_i array can be achieved in-memory. Data access in the memory can also be sequential or random in the binary tree structure.

For large amounts of geospatial field data, it is common that the HGFDR data cannot fit into the primary memory of a single machine. For this purpose, a file-based data structure, which supports streaming access and flexible data storage, is therefore more flexible for large volume data storage. Compared with the in-memory data structures, however, the file-based data structure is more complex. In modern stream-based files, the data can only be stored linearly, thus an index table should first be developed to indicate the start and end location of each U_i and B_i . To achieve better performance, the organization of the U_i and B_i should be accessed with the least traverse cost.

To support out-of-memory data storage, a file-based storage data structure was designed (Fig. 6). First, a file description header, which logs the basic descriptions of the data, was defined. The size of the array stored for each node and its start and end locations were logged. For the real coefficient data storage, each node was stored in a parallel fashion as a separate file or data block. The use of the separate files or separate data blocks depends on the disk I/O. In a single hard disk, the separate files and data block solutions may have similar performance, but for servers with parallel disks, like raid 0 disk arrays, the separate-file solution may have much simpler management and access performance. In each data updating or processing step, the size, start/end location of the node arrays and descriptions of the file are updated synchronously. With this data structure, the data can be accessed as following:

- Data reconstruction. The data reconstruction needs to traverse all the data. Every node is accessed from the leaf node to the root node. The subspace is constructed with a subtree stored sequentially.
- Data query. Since nodes are stored separately, and the start and end location of the data section are logged. Both the location query and range query can access each node and select the subset independently.
- 3) *Data appending and updating*. These data manipulations require accessing each node, first processing the leaf node separately and then modifying the interior node sequentially.

5 EXPERIMENTS

5.1 Data and Configuration

The 2° × 2° 20th Century Reanalysis Monthly Mean Composites Grid (http://www.cdc.noaa.gov/Composites/) from 1871-01 through 2010-12 were selected as the experimental data. The data set have eight attributes: Air Temperature (Air), Geopotential Height (Hig), Zonal Wind (Uwnd), Meridional Wind (Vwnd), Wind Speed (Wspd), Specific Humidity (Shum), Omega, and Relative Humidity (Rhum). For each attribute is stored as a $180 \times 91 \times 24 \times 1,680$ (*latitude* × *longitude* × *pressurelevel* × *time*) tensor. The memory occupation of any single attribute was about 6 GB. By importing the data from the NetCDF into the memory, we have a total amount of 48 GB data for test the performance of our solution.

Three key indices, relative error, peak memory occupation and computation time, are used to benchmark the performances. The selection of the peak memory as a measure is based on the understanding that most computations (e.g., the SVD procedure) usually occupy several times more memory than the original data [36]. The following experiments were performed: 1) simulations with different block sizes and ranks were performed with data of a constant size to find the optimized block size and rank; 2) incremental data appending with the optimized block size and rank to reveal the robustness of the algorithm for continuous appending data; 3) simulations with different buffer sizes with constant block and rank to test the impacts of the buffer size; 4) comparison of the compressed performance between our solution and commonly used scientific data formats; 5) comparison between query and computation performance of our method and exiting tensor approaches.

The main routines of our algorithms are implemented as a plugging-in of the system CAUSTA [37] with Visual C++ 2010 compiler and Intel MKL library. Htucker toolbox 1.2 [28], TT-toolbox 2.2 [21] and the tensor toolbox 2.4 [38] are selected as references for the performance of query and computation evaluation. Since these toolboxes are all implemented with MATLAB, we call our routines in MATLAB with the mex compiler. All the tests were performed in the MATLAB 2011b environment on an Inspur NP 3560 server with two Intel Xeon E5645 (2.4 G) processors and 48 GB DDR-3 ECC Memory.

Fig. 7. Simulation result of different rank and block size.

5.2 Selection of the Parameters

The determination of the optimized rank and block size played a key role in the Blocked-HGFDR. In the configuration, the first 120-month data for each attribute are first selected as the initial data and are represented with different block sizes and ranks. To keep the consistency of the original data, we selected nine common divisors of the size of the temporal dimension of the original data as different block sizes (6, 8, 10, 12, 20, 24, 30, 60, 120). The rank size changes from 40 to 300 with a step of 20. For the initial data, the buffer size is set to zero to avoid the buffer data transfer impact for the performance. The three performance indices were measured and compared for each block size and rank. The situation without blocking is also computed.

The impacts of the rank and block size on the performance indices are similar among all the eight attributes. So, only one typical result of the Air index is illustrated in Fig. 7. The relative error is mainly controlled by the rank *k*. For all the attributes, the relative errors quickly approached to zero after rank k > 160. All simulations have a relative error at the 10^{-6} level, which is acceptable for most engineering tasks. The peak memory occupations and running times are affected by both the block size and the rank. Compared with the original HGFDR without blocking, the blocked-HGFDR requires much less memory and computation time. For the same precision of 10^{-6} level, the original HGFDR requires a rank k > 876 and a peak memory occupations more than 794M for all the eight attributes. However, the blocked version only requires rank k > 160 and peak memory less than 320M. The running time is also reduced from dozens of minutes to less than one minute with the blocking mechanism. For the blocked solution, much smaller rank can achieve the full precision for representation of each block, thus can greatly reduce the memory cost and time occupation.

Because of the recursive use of the SVD in the HGFDR, larger block size and rank will result in more memory and more computation time. For block size smaller than 60, larger block size can accelerate the computation. However, the block size 120 requires more time. This is because the time cost in data aggregation are becoming larger when the block size is too large. With all attributes considered, a block size b = 30 and rank k = 150 are the optimal values for these parameters for achieving a balance between accuracy and performance.

The data appending at the interval of 120 months was simulated for continuous data appending. Each piece of the appending data is represented as HGFDR with block size b = 30 and rank k = 150 and appended to each attribute synchronously and then compressed. For each time, four blocks of each attribute are appending to the existing data and then compressed. The relative error, cumulative time

Fig. 8. Continues updating with different blocks and buffer.

and peak memory usage for each data appending are shown in Fig. 8a. Clearly, the peak memory is very stable at the level of 90M for every data appending. Since the data appending is performed to each block individually, the data accumulation will not affect the memory cost for each data appending. The accumulated compression error, although it has a growing trend, is still at the 10^{-6} level for all the time. Because the appending data amount is equal for every update, the cumulative time grows linearly. This suggests the time cost for each data updating is stable and only affected by the amount of data currently appended. All these results suggest the performance of our method is stable for continuous data appending.

The buffer size affects the computation balance between the compression and data updating. Simulation with different buffer sizes is also tested with the data. Since the memory requirement of each node is not consistent, it is not suitable to provide a constant buffer size for all the nodes. Here, we use nodes number stored in the buffer (buffer number) as the index for the buffer size. The performance indices change with the buffer number increasing from 0 to 11 (Fig. 8b). Bigger buffer size will lead to less time of data compression, which has very small impact on the data precision. Therefore, bigger buffer size will lead to more accurate data representation. However, when the buffer number increases, the peak memory requirement and computation time increase rapidly. Both the time and memory requirement grow rapidly when the buffer number is bigger than 5.

5.3 Performance Comparison

The compression ratio is used to compare HGFDR with most commonly used data formats (Table 1). The file of each attribute is stored with the ASCII(ASC), GeoTIFF, MATLAB Binary file(.mat v7.3), NetCDF and HDF formats. The ASC format and GeoTIFF can be seemed as the uncompressed and the others can be seen as compressed data formats. The HGFDR is repented with rank k = 150 and block

TABLE 1 File Size of Different Formats (Unit:GB)

	ASC	GeoTIFF	.Mat	NetCDF	HDF	HGFDR
Air	5.32	4.95	1.09	1.22	1.23	0.53
Hgt	5.32	4.03	1.09	1.22	0.99	0.57
Omega	5.32	4.02	1.08	0.97	0.99	0.32
Rhum	5.32	4.01	1.08	0.97	0.99	0.34
Shum	5.32	4.02	1.08	0.97	0.99	0.31
Uwnd	5.32	4.91	1.09	1.22	1.22	0.61
Vwnd	5.32	4.92	1.09	1.22	1.22	0.66
Wspd	5.32	4.92	1.09	1.22	1.23	0.68

Fig. 9. Representation performance comparison.

size b = 30. Clearly, the HGFDR had the lowest cost in terms of space requirement among all the file formats. Compared with the non-compressed formats such as ASC and Geo-TIFF, the HGFDR only requires less than 10 percent storage space. Even compared with the commonly used compressed data formats, the HGFDR can reduce half space required. The compression ratios between different attributes also reveal the effect of the structural complexity on them. The indices for Omega, Rhum and Shum, which have simpler structures, had much higher compression ratios. The Wspd data, composed from the latitude and longitude vector wind data field, had the lowest ratio.

To compare the appending, querying and computation performances with exiting tensor solutions, the data are also represented as two commonly used hierarchical tensor formats: htucker [28] and Tensor-Train (TT)[21]. The similar data appending is also performed by appending the block data into the existing data. The performance comparison is depicted in Fig. 9. All three have the relative error at the level 10^{-6} , and the error distributions are very similar. Our method and htucker method have much less time cost than the TT format, especially when the data volume gets big. Due to the cost of the updating of the blocks, the time cost of our solution is a little bit higher than the htucker solution when the blocks are growing. However, for the memory usage, our solution definitely requires much less memory than the other two. The memory cost accumulation is stable when the volume of the data appended is fixed. This property makes the blocked-HGFDR suitable for massive volumes of high dimensional data storage and updating.

The querying performances of HGFDR are also evaluated and compared (Fig. 10a). Different subset percentages are selected by indexing each dimension randomly from the data represented as HGFDR, TT, original Tensor and MAT-LAB multidimensional array formats. Clearly, our solution have the highest performance among all these solutions. Since the data structure of the HGFDR is in fact a search tree and the queries of the multidimensional data are flattened in a hierarchical structure that is independent of dimensions. No data reformatting or combining operators

Fig. 10. Query and computation performance.

are needed. Even for the random query for 80 percent of original data, the HGFDR only requires less than 0.11 second. However, the classical matrix and tensor approach requires more than 1.24 second. The TT method cost most time for random query, which may be caused by the reconstruction from the long chains of matrix computation.

The HGFDR representation can be used not only for data compression, updating and query, but also to support the computation with linear tensor operators [28]. Since the computation performance provides an important perspective on the HGFDR-based analysis workflow, we also tested the performance of the HGFDR according to [28]. As shown Fig. 10b, the experiments were developed to compare the computation of the Fast Fourier Transformation (FFT) of a tensor. The computation was performed by the native MAT-LAB function fft(), applied for both matrix and the HGFDR format (only the air variable with compressed rank k = 250and block size b = 30 is illustrated here). The memory usage and computation time were computed and compared. The accuracy of the Fast Fourier Transformation was also tested by the inverse FFT of the resulting tensor and the reconstruction error was computed. Since the linear operators applied to the HGFDR can be split and applied to each node, both the memory usage and computation time are much less than the original matrix computation. The reconstruction errors are no higher than 10^{-6} (Fig. 10). Because the HGFDR form, both in-memory and with the file-based HGFDR representation, provides highly efficient and compact data storage, it is possible to construct an integrated data analysis workflow that has high efficiency and a unified data structure.

6 CONCLUSIONS

Geoscience research is shifting to a data-driven stage. New computational tools and data intensive scalable architectures that can support unified storage, query and complex analysis for such massive multidimensional datasets will be critical [37]. Tensor is a natural way of representing multidimensional field data. In this paper, the HGFDR, which can support the process of representation, updating, compression, query and analysis on massive multidimensional geospatial field data, was proposed. By the split-and-merge paradigm, the HGFDR achieves the balance between data accuracy, memory occupation and running time for such data. The compression ratio can be customized according to either the data scales or the accuracy. Continuous data appending and compression allow the data scale to be controlled at particular levels without losing the accuracy of data representation. The query, and many of the commonly used analysis methods, can be accelerated with the blocked-HGFDR. The experiments suggest that our method provides an effective tool for multidimensional data representation and analysis. Since the computational efficiency is very high and the memory cost is low even with high volumes of data, our method has the potential for processing massive amounts of data on a single PC.

The HGFDR allows a computational engine, including the data management operators and data analysis operators, to be constructed. Since it can be easily updated, queried, computed and compressed without changing its structure, HGFDR can be easily used to form analysis workflow, including data operation and analysis operators, which can be constructed and optimized algebraically. With the help of the dimensional independent property of the HGFDR, some of the workflows can be computed in parallel, which can additionally improve the computational performance. The data analysis operators, such as vector/matrix/tensor products, and addition and complex operators such as FFT and SVD, can be used to form complex analysis workflow, since both the data operation and the analysis operators do not change the structure of the HGFDR representation. The integration of the total data representation, storage, operation and workflow analysis makes the HGFDR an important contribution for productive applications.

Our future works include: 1) determine the strategy for finding best block splitting and rank determination according to the data distribution; 2) develop the parallel data query and analysis operators and algorithms to further increase the scalability and performance; 3) integrate more advanced analysis tools, such as multidimensional calculus, ordinary differential equations and partial differential equations, with the help of tensor structure, which could greatly improve the computational ability of the HGFDR representation.

ACKNOWLEDGMENTS

This work was supported by the NSFC project (Grant No. 41201377, 41231173), the NCET program (Grant No. NCET-12-0735), and the PAPD project of Jiangsu Higher Education Institutions. L. Yuan is the corresponding author.

REFERENCES

- H. B. Newman, M. H. Ellisman, and J. A. Orcutt, "Data-intensive e-science frontier research," *Commun. ACM*, vol. 46, no. 11, pp. 68–77, 2003.
- [2] M. A. Wulder, J. G. Masek, W. B. Cohen, T. R. Loveland, and C. E. Woodcock, "Opening the archive: How free data has enabled the science and monitoring promise of landsat," *Remote Sens. Environ.*, vol. 122, pp. 2–10, 2012.
- [3] M. Gertz, Q. Hart, C. Rueda, S. Singhal, and J. Zhang, "A data and query model for streaming geospatial image data," in *Proc. Cur*rent Trends Database Technol., 2006, pp. 687–699.
- [4] T. L. L. Siqueira, C. D. de Aguiar Ciferri, V. C. Times, and R. R. Ciferri, "The sb-index and the hsb-index: Efficient indices for spatial data warehouses," *Geoinformat.*, vol. 16, no. 1, pp. 165–205, 2012.
 [5] M. L. Yiu, H. Lu, N. Mamoulis, and M. Vaitis, "Ranking spatial
- [5] M. L. Yiu, H. Lu, N. Mamoulis, and M. Vaitis, "Ranking spatial data by quality preferences," *IEEE Trans. Knowl. Data Eng.*, vol. 23, no. 3, pp. 433–446, Mar. 2011.
- [6] A. Aji, F. Wang, H. Vo, R. Lee, Q. Liu, X. Zhang, and J. Saltz, "Hadoop gis: A high performance spatial data warehousing system over mapreduce," *Proc. VLDB Endowment*, vol. 6, no. 11, pp. 1009–1020, 2013.
- [7] C. Reed, ⁷14 OGC standards and geospatial big data," in *Big Data: Techniques and Technologies in Geoinformatics*, Boca Raton, FL, USA: CRC Press, 2014, pp. 279–289.
- [8] E. van der Zee and H. Scholten, "Spatial dimensions of big data: Application of geographical concepts and spatial technology to the internet of things," in *Big Data and Internet of Things: A Roadmap for Smart Environments*. New York, NY, USA: Springer, 2014, pp. 137–168.
- [9] S. J. Kazemitabar, F. Banaei-Kashani, and D. McLeod, "Geostreaming in cloud," in *Proc. 2nd ACM SIGSPATIAL Int. Work-shop GeoStreaming*, 2011, pp. 3–9.
- [10] G. Cugola and A. Margara, "Processing flows of information: From data stream to complex event processing," ACM Comput. Surv., vol. 44, no. 3, pp. 15:1–15:62, 2012.

- [11] R. J. Harrison, B. Bader, L.-h. Lim, C. Ding, L. Eld, C. Martin, L. Mullin, S. P. Ponnapalli, and P. Regalia, "Future directions in tensor-based computation and modeling," Workshop Report in Arlington, Virginia at National Science Foundation, 2009, pp. 1–20. Available: http://www.cs.cornell.edu/cv/TenWork/ Home.htm
- [12] F. Vogt, J. Cramer, and K. Booksh, "Introducing multidimensional hybrid wavelets for enhanced evaluation of hyperspectral image cubes and multi-way data sets," J. Chemometrics, vol. 19, no. 9, pp. 510–520, 2005.
- [13] H. Wang and N. Ahuja, "Compact representation of multidimensional data using tensor rank-one decomposition," *Pattern Recog. Int. Conf. IEEE Comput. Society*, vol. 1, pp. 44–47, 2004.
- [14] M. Ishteva, L. De Lathauwer, P. Absil, and S. Van Huffel, "Dimensionality reduction for higher-order tensors: Algorithms and applications," *Int. J. Pure Appl. Math.*, vol. 42, no. 3, pp. 337–343, 2008.
- [15] Q. Wu, T. Xia, C. Chen, H.-Y. S. Lin, H. Wang, and Y. Yu, "Hierarchical tensor approximation of multi-dimensional visual date," *IEEE Trans. Vis. Comput. Graphics*, vol. 14, no. 1, pp. 186–199, Jan./Feb. 2008.
- [16] Z. Yu, L. Yuan, G. Lü, W. Luo, and Z. Xie, "Coupling characteristics of zonal and meridional sea level change revealed by satellite altimetry data and their response to ENSO events," *Chinese J. Geophys.*, vol. 54, no. 8, pp. 1972–1982, 2011.
- *Chinese J. Geophys.*, vol. 54, no. 8, pp. 1972–1982, 2011.
 [17] Z. Yu, L. Yuan, W. Luo, L. Yi, and J. Wang, "Tensor-based topographical spatial-temporal field data organization and analysis," *J. Remote Sens. Technol. Appl.*, vol. 27, no. 5, pp. 699–705, 2012.
- [18] W. Luo, L. Yuan, Z. Yu, and et al, "Principle tensor-based spatialtemporal structure analysis and dynamical expression," J. Bas. Sci. Eng., vol. 21, no. 2, pp. 276–286, 2013.
- [19] I. V. Oseledets and E. E. Tyrtyshnikov, "Breaking the curse of dimensionality, or how to use SVD in many dimensions," SIAM J. Sci. Comput., vol. 31, no. 5, pp. 3744–3759, 2009.
- [20] C. Böhm, S. Berchtold, and D. A. Keim, "Searching in highdimensional spaces: Index structures for improving the performance of multimedia databases," ACM Comput. Surv., vol. 33, no. 3, pp. 322–373, 2001.
- [21] I. Oseledets and E. Tyrtyshnikov, "TT-cross approximation for multidimensional arrays," *Linear Algebra Appl.*, vol. 432, no. 1, pp. 70–88, 2010.
- [22] L. Grasedyck, "Hierarchical singular value decomposition of tensors," SIAM J. Matrix Anal. Appl., vol. 31, no. 4, pp. 2029–2054, 2010.
- [23] U. Benedikt, A. Auer, M. Espig, and W. Hackbusch, "Tensor decomposition in post-Hartree Fock methods. Pt. 1 : Two-electron integrals and MP2," J. Chemical Phys., vol. 134, no. 5, p. 054118, 2011.
- [24] I. Arad and Z. Landau, "Quantum computation and the evaluation of tensor networks," SIAM J. Comput., vol. 39, no. 7, pp. 3089– 3121, 2010.
- [25] J. Salmi, A. Richter, and V. Koivunen, "Sequential unfolding SVD for tensors with applications in array signal processing," *IEEE Trans. Signal Process.*, vol. 57, no. 12, pp. 4719–4733, Dec. 2009.
- [26] C. Lubich, T. Rohwedder, R. Schneider, and B. Vandereycken, "Dynamical approximation by hierarchical tucker and tensortrain tensors," *SIAM J. Matrix Anal. Appl.*, vol. 34, no. 2, pp. 470– 494, 2013.
- [27] L. Grasedyck, D. Kressner, and C. Tobler, "A literature survey of low-rank tensor approximation techniques," *GAMM-Mitteilungen*, vol. 36, no. 1, pp. 53–78, 2013.
- [28] D. Kressner and C. Tobler, "Algorithm 941: Htucker-a matlab toolbox for tensors in hierarchical tucker format," ACM Trans. Math. Softw., vol. 40, no. 3, p. 22, 2014.
- [29] M. Espig, M. Schuster, A. Killaitis, N. Waldren, P. Wähnert, S. Handschuh, and H. Auer. (2012). Tensorcalculus library [Online]. Available at http://gitorious.org/tensorcalculus
- [30] H.-P. Kriegel, P. Kröger, and A. Zimek, "Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering," ACM Trans. Knowl. Discovery Data, vol. 3, no. 1, pp. 1:1–1:58, 2009.
- [31] N. Karayannidis and T. Sellis, "Hierarchical clustering for olap: The cube file approach," VLDB J. Int. J. Very Large Data Bases, vol. 17, no. 4, pp. 621–655, 2008.
- [32] X. Guan, H. Wu, and L. Li, "A parallel framework for processing massive spatial data with a split-and-merge paradigm," *Trans. GIS*, vol. 16, no. 6, pp. 829–843, 2012.

- [33] L. Arge, "The buffer tree: A technique for designing batched external data structures," *Algorithmica*, vol. 37, no. 1, pp. 1–24, 2003.
- [34] L. L. Scharf, Statistical Signal Processing, vol. 98, Reading, MA, USA: Addison-Wesley, 1991.
- [35] H. Plattner and A. Zeier, In-Memory Data Management: An Inflection Point for Enterprise Applications. New York, NY, USA: Springer, 2011.
- [36] T. G. Kolda and J. Sun, "Scalable tensor decompositions for multiaspect data mining," in *Proc. IEEE 8th Int. Conf. Data Mining*, Dec. 2008, pp. 363–372.
- [37] L. Yuan, Z. Yu, S. Chen, W. Luo, Y. Wang, and G. Lü, "CAUSTA: Clifford algebra-based unified spatio-temporal analysis," *Trans. GIS*, vol. 14, no. s1, pp. 59–83, 2010.
 [38] B. W. Bader and T. G. Kolda. (2010, Mar.). Matlab tensor toolbox
- [38] B. W. Bader and T. G. Kolda. (2010, Mar.). Matlab tensor toolbox version 2.4 [Online]. Available: http://csmr.ca.sandia.gov/~ tgkolda/TensorToolbox

Linwang Yuan received the BS degree in geography science, the MS and PhD degrees in geographical information system from Nanjing Normal University, China, in 1995, 1998, and 2001, respectively. He is currently a professor of the Key Laboratory of VGE, Ministry of Education. His research interests include geographic information science and geographical modeling.

Zhaoyuan Yu received the BS degree in geography science, and PhD degrees in GIS and cartography from Nanjing Normal University, China, in 2005 and 2010, respectively. He is currently a lecturer in the Geography Department at Nanjing Normal University, China. His research interests include high-dimensional geophysical signal processing and geographical information system.

Wen Luo received the BS degree in geography science from Hubei University, Wuhan, China, in 2008, the MS degree in geographical information system from Nanjing Normal University, China, where he has been working toward the PhD degrees since 2011 and majored in the field of geographical information system. His research interests include geographical information system, geometric algebra and computer algorithms.

Yong Hu received the BS degree in computer science and the MS degree in operation and control theory both from Nanjing University, China, in 1995 and 2004, respectively. He is currently a lecturer in the Computer Science and Technology Department at Nanjing Normal University. His research interests include big data and cloud computing, computer algorithms, and system architecture.

Linyao Feng received the BS degree in geographical information system from Southwest University, Sichuan, China, in 2012, he has been working toward the MS degree since 2011 and majored in the field of geographical information system. Her research interests include application of remote sensing, geographical information system, and geographical modeling.

Axing Zhu received the BS degree in geography from Beijing Normal University, China, in 1983, the MSc degree in geography from the University of Calgary, Canada, in 1987, and the PhD degree from the University of Toronto, Canada. His research interests include geographical information systems and remote sensing techniques, and their application in environmental modeling and natural resource management.

▷ For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/publications/dlib.