A Multi-dimensional Data Storage Using Quad-tree and Z-Ordering

Fang Hou, Chenghui Huang, Jiyuan Lu
Department of Computer Science and Technology
Guangdong University of Finance
Guangzhou, China
e-mail: hfhoufang@tom.com

***Abstract*—Multi-dimensional applications use tree structure to store data and space filling curves to traverse data. Most frequently used Quad-tree and Z-ordering curve are analyzed. By importing these to a HDF5 file format, a multi-dimensional data storage subsystem is constructed. Performance test results show in a sequential reading application environment, this method is feasible and efficient.**

Keywords-storage, multi-dimensional, performance, data structure,

I. INTRODUCTION

A multi-dimensional data processing procedure includes data generation, data (In-Situ)[1] compression and decompression, data storage, data query, data transport, data simulation, etc. A lot of High Performance Computer (HPC) systems have been implemented to process multi-dimensional data in scientific research, industry and business area.

A typical storage capacity of a HPC system exceeds a scale of ExaBytes (2^{60}bytes). The fundamental function of storage subsystem is to load/store data to/from main memory. IBM Sequoia Blue Gene/Q, the newest No.1 HPC system on the TOP500 list published on June 2012[2], owns a 1,572,864 GB main memory(Top500org 2012). Scott Klasky[3] defined an effective speed index of storage system to evaluate the I/O speed for HPC systems. This index equals the time of writing system's whole main memory to file system. For instance, the index of the ASCI purple system is 49 TB/140 GB/s (=358.4s), and this of the Jaguar system is 300 TB/200 GB/s (=1536s). Although the bandwidth of storage subsystem has been increasing steadily, transferring whole main memory to storage system takes more and more time, especially all the HPC systems prefer to an increasing mass main memory capacity.

In terms of random accesses for multi-dimensional data, the response latency of hard disks based storage system is about 100ms. Fortunately, multi-dimensional applications rarely perform arbitrarily random accesses. For example, although a user may navigate a simulated scene in a random direction in a 3-D virtual reality situation, data needed to render new images are located in space unites which are adjacent to the current space unit. This feature makes cache technology valuable.

To narrowing the performance gap between data process subsystem and data storage subsystem, researchers have dedicated to data access pattern, data structure, file systems and formats, and I/O optimization for recent years. The following of this note will summarize these results respectively.

II. CHARACTERISTICS OF MULTI-DIMENSIONAL DATA ACCESS

Multi-dimensional data of applications is accessed generally in 6 patterns[3]:

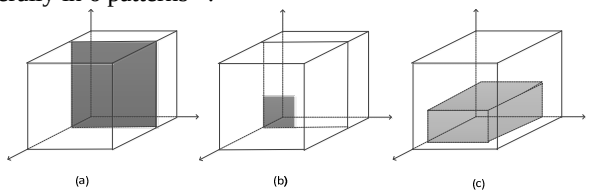

Figure 1 Data area accessed by different patterns

(1) Separated write and read pattern. Data is created by one or more process, and is retrieved by one or more process. For example, a 3-D data set is created by sampling program or simulation program, and then was compressed by an In-Situ process. Then this data set is accessed by a query process or a virtual reality process.

(2) Specific parameters access pattern. Several dimensions of all the stored data is retrieved. For instance, a query for all the 3-D velocity is executed to trace the particle movements.

(3) Specific parameter access pattern. Only one dimension of all the stored data is retrieved. For example, a query only for the temperature of the position in 3-D space is executed.

(4) Specific plane access pattern. The multi-dimensional data set is sliced to several planes. A process accesses all the data on one specific plane. See Figure 1(a).

(5) Specific sub plane access pattern. The multi-dimensional data set is sliced to several planes. A process accesses the data belongs to part of this plane. For example, a rectangle area of this plane. See Figure 1 (b).

(6) Specific sub cubic access pattern. The data set is divided into several sub-cubes. A process accesses all the data inside one specific cube. See Figure 1 (c).

III. DATA STRUCTURE AND SPACE FILLING CURVE

A. *Quad-tree data structure*

Several data structures of multi-dimensional application was illustrated in Gaede's survey[4]. Geade's paper divided data structure forms into 2 categories: 1) Basic data structure. The paper introduced several kinds of trees by which multi-

dimensional data was organized in main memory. 2) Point access methods. The paper listed some kinds of hashing file and trees by which multi-dimensional data was stored in secondary storage device.

In main memory, all the tree structures attached importance to depth of specific leaf node search, time complexity of insertion or deletion operations. Besides aforementioned factors, the point access methods took data layout into consideration. Because applications have almost the same time of accesses to any unit in main memory, but time of accesses to different disk sectors varies within a large range.

But most widely accepted multi-dimensional tree structures have already taken disk accesses into consideration. For example, an R*-tree[5] modified R-tree's[6] insertion algorithm to achieve better disk access performance. So, the tree structures in this note will not be categorized in this way.

The most common used data structures for multi-dimensional application are the following 7 kinds of trees: k-d-tree[7], BSP-tree[8], R-tree, R+-tree[9], R*-tree, Quad-tree[10], and Oct-tree[11]. The first 2 are binary trees, the branches numbers of following 3 are not fixed, and the last 2 are configured by dimension numbers of the data objects.

Generally, a tree consists of non-leaf nodes and leaf nodes. In multi-dimensional applications, a non-leaf node can be expressed by: (*I, Child-pointer*), and leaf node can be expressed by: (*I, Tuple-identifier*). In both nodes, the I is the bounding box of a specific multi-dimensional object. For instance, in a 2-D case, $I=\{x_{[a,b]}, y_{[c,d]}\}$ means the data object is in a closed area within [*a, b*] on x-axis, and [*c, d*] on y-axis. In some cases, the term MBR (Minimum Boundary Rectangle) is used to refer this *I*. The *Child-pointer* points to non-leaf node, and the *Tuple-identifier* indicates the location where a specific data object stores.

The Quad-tree recursively split a rectangle area into 4 parts. All parts are of same size. Figure 2(a) shows the whole body is divided into NE, SE, SW, and NW parts, and the NW can be split into another smaller NE, SE, SW and NW parts. Figure 2(b) is the formed region Quad-tree.

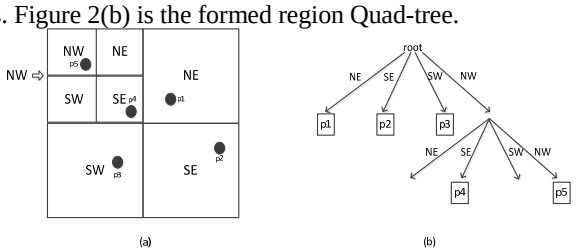

Figure 2 An example of Quad-tree

B. *Z-ordering Space filling curves*

To map multi-dimensional data objects to 1-D objects, several space filling curves have been designed. Most of them are continuous self-similar fractal curves.

There are 2 curves are widely accepted as spacing filling methods for present multi-dimensional applications: Hilbert Curve and Z-Ordering Curve.

(1) Hilbert Curve:

As its name indicates, this curve was proposed by David Hilbert in 19th century. Figure 3(a) is a 2-D Hilbert Curve and (b) is a 3-D curve. The innermost black line in Figure 3(a) is a first order Hilbert curve, the middle black line is a second order one, and the gray line is a third order one. It illustrates how a Hilbert curve can recursively explore a space. By these lines, the particles or rectangles in this 2-D space can be visited in a linear order. It makes mapping 2-D objects to 1-D objects possible. There are mature algorithms for finding out the Hilbert order which are important to perform operations such as searching, deleting and inserting.

The Hilbert Curve combined with an R-tree data structure[12] has been widely adopted in multi-dimensional applications.

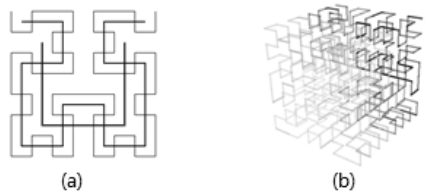

Figure 3 Hilbert Curve in 2-D and 3-D

(2) Z-Ordering Curve:

Z-Ordering curve was introduced by Morten(Morton 1966) in 1966. The z-value of a point in multi-dimensional space is simply calculated by interleaving the binary representations of its coordinate values. Figure 4 shows Z-ordering curves in 2-D and 3-D situations.

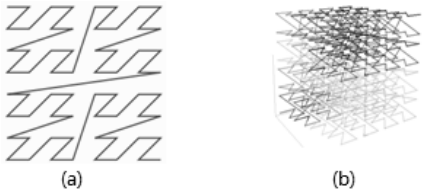

Figure 4 Z-Ordering Curve in 2-D and 3-D

Once the data are sorted into this ordering, any one-dimensional data structure can be used such as binary search trees, B-trees, skip lists or (with low significant bits truncated) hash tables.

IV. AN EXAMPLE OF Z-ORDERING QUAD-TREE STORAGE

A. *Data Structure*

The resulting ordering can equivalently be described as the order one would get from a depth-first traversal of a Quad-tree. Because of its close connection with the Quad-tree, the Z-ordering can be used to efficiently construct a Quad-tree and related higher dimensional data structures. Some detailed information is given in Section 3.6 of this note.

The order of how to store these 4 sub-areas has a tight connection with the space filling curve. The above structure suits a Hilbert curve very well. If a multi-dimensional application adopts a Z-ordering curve, constructing a Quad-tree with the order of SE, SW, NE and NW will be more efficient for data accesses. Finkel[13] suggested that the sub-areas could be unequal to decrease average search depth. All the objects were ordered by their x coordinates primarily and by y coordinates secondly. Then a more balanced tree could

be created. This made the differences between the objects number in each sub-area minimized. Consequently, creation, insertion and deletion of such a Quad-tree are more complicated.

Balmelli[14] incorporated Quad-tree with Z-ordering curve. Figure 7(a) shows a 2-D region recursively divided by Z-ordering. The rectangle with shadow means there is objects in this area. Figure 7(b) shows how these nodes are stored into a Z-ordering Quad-tree.

B. Data node organization

Hierarchical data format version 5 (HDF5)[15] is managed by the University of Illinois, adopted by several laboratories such as National Aeronautics and Space Administration (NASA), National Science Foundation (NSF), Department of Energy (DOE), etc. Today, it has been formally adopted by the International Standard Organization (ISO). In the Part 26 of its specification 10303 (Standard for the Exchange of Product model data, STEP), HDF5 is the standard format for the binary representation of EXPRSS-driven data.

A HDF5 consists of *groups* and *datasets*. Figure 8 shows the structure of HDF5. The *groups* describes the data organization structure, while the *datasets* offers basic data storage structure. Innately, the *datasets* is designed for multi-dimensional array of elements. The *data space* area defines the dimensions of data, and the *contiguous* type of *storage layout* consorts the multi-dimensional data stream. Furthermore, the *chunked* method brings 3 advantages to multi-dimensional data storage:

- a specific subarea of the whole data set can be stored into a chunk. That means the data can be organized by chunks;
- algorithms of compressing and decompressing can be applied on the data in a chunk;
- dimension extending or reducing can be conducted by chunk units.

The most attractive feature of HDF for a multi-dimensional application is that the data can be stored into the storage devices by a contiguous way, which means a direct map from physical memory, or by a chunked way. The latter is an equal-size structured node way.

C. Performance results

Using Z-ordering Quad-tree structure in a HDF5 file format, our research implemented a multi-dimensional application for a multicellular metal fluid flow simulation.

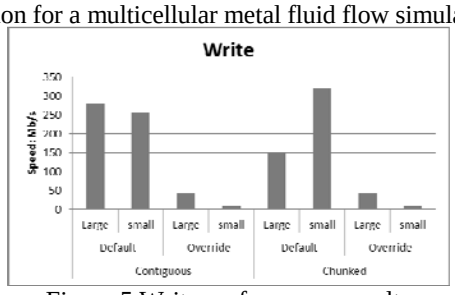

Figure 5 Write performance results

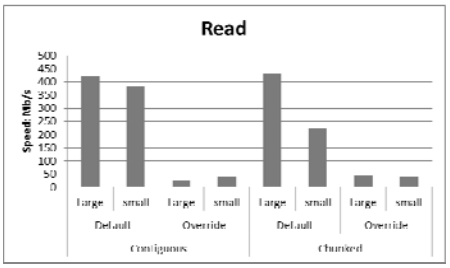

Figure 6 Read performance results

Figure 5 and Figure 6 shows the performance results of write and read test.

Contiguous means the data in storage is in the same linear continuous sequence as in memory. *Chunked* means the data is organized in same sized units. *Default* and *Override* refer to if the system cache was flushed before test. *Large* and *Small* illustrate size of each dimension.

V. CONCLUSION

Most of tree structure algorithm optimizations are application sensitive. At the same time, most of multi-dimensional applications focus on sequential reading. Objects searching operations will be performed more frequently than objects insertions or deletions. For example, since many program uses a fixed size unit of 16*16*16, the Z-ordering-Quad tree is an appropriate choice for data structure.

The organization of data nodes depends on which kind of multi-dimensional data accesses the application performs. In a sequential read/write case, the HDF5 will be more suitable.

ACKNOWLEDGMENT

This work is Supported by the Natural Science Foundation of Guangdong Province, China (Grant No.S2012040007847)

REFERENCES

[1] Scott Klasky, Hasan Abbasi, Jeremy Logan, et al. In situ data processing for extreme scale computing [C]. Scientific Discovery through Advanced Computing Program (SciDAC), Denver, Colorado. 2011:

[2] Top500org (2012). Retrieved 10.29, 2012., from http://i.top500.org/system/177556.

[3] Jay Lofstead, Milo Polte, Garth Gibson, et al. Six degrees of scientific data: reading patterns for extreme scale science IO[C]. Proceedings of the 20th international symposium on High performance distributed computing, San Jose, California, USA, ACM. 2011:49-60.

[4] Volker Gaede,Oliver Guenther. Multidimensional access methods[J]. ACM Computing Surveys. 1998, 30(2): 170-231.

[5] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, et al. The R*-tree: an efficient and robust access method for points and rectangles[J]. SIGMOD Rec. 1990, 19(2): 322-331.

[6] Antonin Guttman. R-trees: a dynamic index structure for spatial searching[J]. SIGMOD Rec. 1984, **14**(2): 47-57.

[7] Jon Louis Bentley. Multidimensional binary search trees used for associative searching[J]. Commun. ACM. 1975, 18(9): 509-517.

[8] Henry Fuchs, Zvi M. Kedem,Bruce F. Naylor. On visible surface generation by a priori tree structures[J]. SIGGRAPH Comput. Graph. 1980, 14(3): 124-133.

[9] Timos K. Sellis, Nick Roussopoulos,Christos Faloutsos. The R+-Tree: A Dynamic Index for Multi-Dimensional Objects[C]. Proceedings of

the 13th International Conference on Very Large Data Bases, Morgan Kaufmann Publishers Inc. 1987:507-518.

[10] R. A. Finkel,J. L. Bentley. Quad trees a data structure for retrieval on composite keys[J]. Acta Informatica. 1974, **4**(1): 1-9.

[11] H SAMET. The quadtree and related hierarchical data structure.[J]. ACM Computer Survey. 1984, **16**(2): 187-260.

[12] Ibrahim Kamel,Christos Faloutsos. Hilbert R-Tree: An Improved R-Tree Using Fractals. ISR; TR 1993-19.

[13] Finkel, R. A. and J. L. Bentley (1974). "Quad trees a data structure for retrieval on composite keys." Acta Informatica 4(1): 1-9.

[14] L. Balmelli, J Kovacevic,M. Vetterli. Quadtrees for embedded surface visualization: constraints and efficient data structures[C]. 1999 International Conference on Image Processing, ICIP 99, KOBE, JAPAN, IEEE Society. 1999:487-491, vol.482

[15] Matthew T. Dougherty, Michael J. Folk, Erez Zadok, et al. Unifying biological image formats with HDF5[J]. Communications of ACM. 2009, **52**(10): 42-47.

.

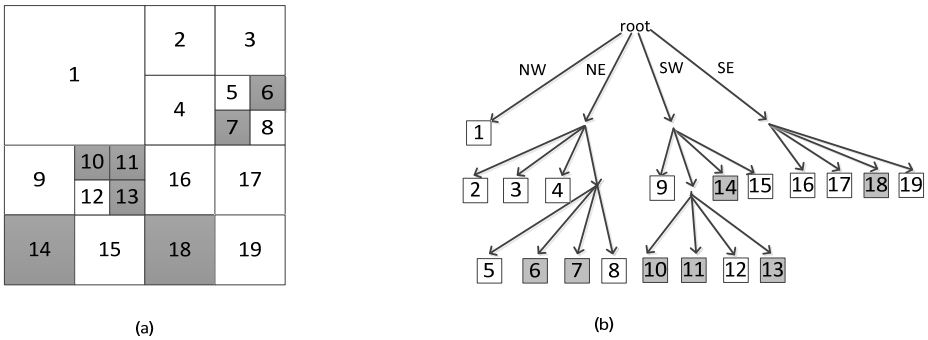

Figure 7 An example of Z-ordering Quad-tree

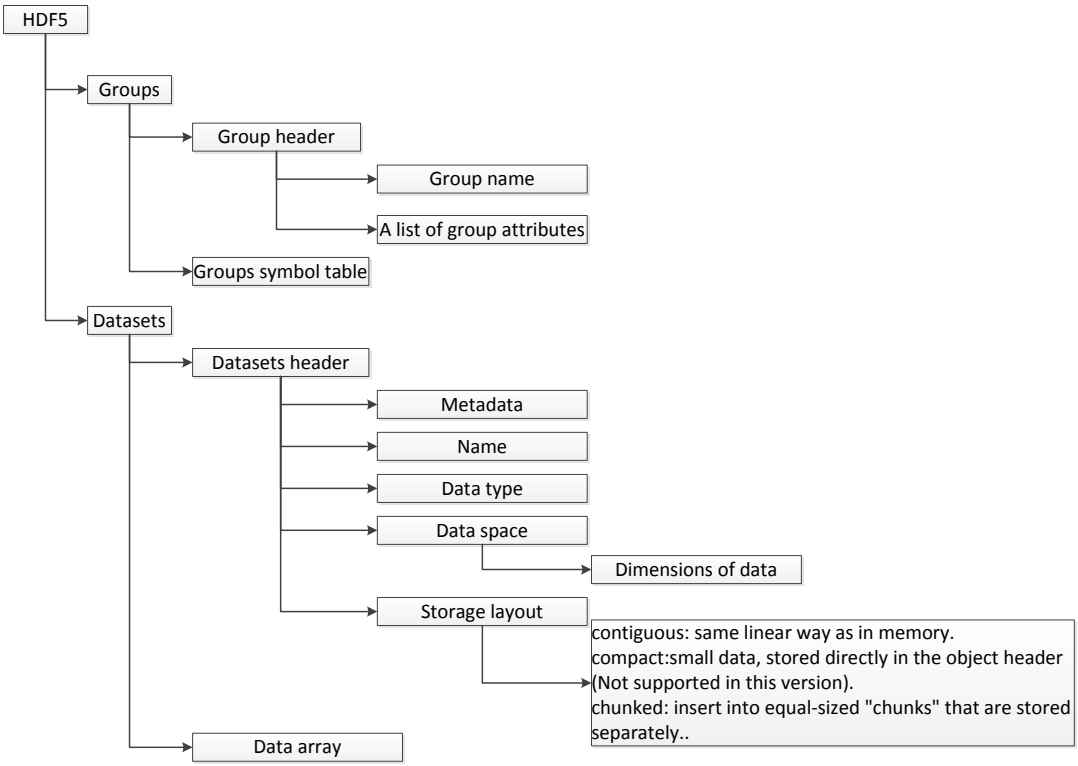

Figure 8 HDF5 file node for multi-dimensional data