

# Multilevel Chunking of Multidimensional Arrays

Arsany S. Sawires  
Dept of Computer Science  
Faculty of Engineering,  
Alexandria University, Egypt  
a.sawires@menanet.net

Nagwa M. El Makky  
Dept of Computer Science  
Faculty of Engineering,  
Alexandria University, Egypt  
elmakky@acm.org

Khalil M. Ahmed  
Dept of Computer Science  
Faculty of Engineering,  
Alexandria University, Egypt  
kmaaaa@hotmail.com

## Abstract

*Multidimensional arrays have become very common in scientific and business applications. Such arrays are usually very large and hence they are stored on secondary or tertiary storage systems. For processing, various parts of a multidimensional array are typically retrieved to main memory using range queries. The performance of such queries is significantly affected by the physical organization of the target array on the storage system. Previous research has proposed an approach of this physical organization by chunking, i.e. by decomposing a multidimensional array into smaller multidimensional arrays (chunks). The parameters of the chunking process are chosen based on the storage system characteristics as well as the expected access pattern of range queries. With this approach, the average time of answering a query has been improved. The research presented in this paper extends the basic approach of chunking to a multilevel version; hence the presented approach is called Multilevel Chunking. We show by experiment that multilevel chunking outperforms single level chunking. The approach presented here primarily targets hard disk systems.*

## 1. Introduction

Multidimensional data views have become essential for many applications. Multidimensional arrays are usually very large and they are stored on secondary or tertiary storage systems. The main design goal in a typical application is to achieve high efficiency for queries that access the array. In many cases, the entire array is stored in some linear order on the storage system and various parts of the array are then retrieved using axes-aligned range queries.

Clustering is always a desired quality when dealing with non-random-access systems such as hard disks or magnetic tapes. Well-clustered data can be retrieved by accessing a small number of I/O pages. Obviously, the linear order chosen to store the array cells significantly affects the clustering quality. Hence, the question is: what is the best linear order to use for the physical organization of the array?

Traditional systems linearize arrays by a nested traversal order. For the case of 2D arrays, two nested traversal alternatives exist: the row major order and the column major order. For an  $n$ -dimensional array, there are

$n!$  alternatives corresponding to the  $n!$  ways of reordering the dimensions. Over time, a large family of linear ordering approaches was discovered; the nested traversal approach has been seen as a member of this family. It is the family of space filling curves (fractals) [1] e.g. the Peano curve, the Hilbert curve, and the Sweep curve which is the traditional nested traversal order. These linearization techniques have been used for indexing multidimensional arrays. They can also be used to solve the clustering problem mentioned above. These techniques, however, do not take into account the different characteristics of the different storage systems. Also, they do not adapt to the different query access patterns. Hence, their clustering quality is independent of the characteristics of the storage system and the expected access pattern. Clearly, a technique that can make use of any information about these characteristics is likely to perform better.

Previous work [2] presented a physical organization approach that takes into account both the factors of (1) The storage system characteristics and (2) The query access pattern. This approach, which is called Chunking, calls for decomposing the multidimensional array into smaller multidimensional arrays (chunks) and storing each chunk on a disk block. The parameters of the chunking process, namely the chunk size and shape, are determined based on the two factors mentioned above. It was shown by experiment that the chunking approach outperforms the traditional nested traversal in terms of the average I/O time per query.

In this paper we propose a Multilevel Chunking approach which extends the basic chunking approach by applying it at several levels rather than at a single level. The chunking parameters are determined based on the physical characteristics of the storage system as well as the expected access pattern of range queries. The experimental results show that Multilevel Chunking outperforms single-level chunking in terms of the average I/O time per query.

The rest of this paper is organized as follows: section 2 summarizes the basic chunking approach. Section 3 describes the proposed Multilevel Chunking strategy for hard disks. Section 4 formalizes the problem and the proposed strategy. Finally, section 5 presents the experimental results.

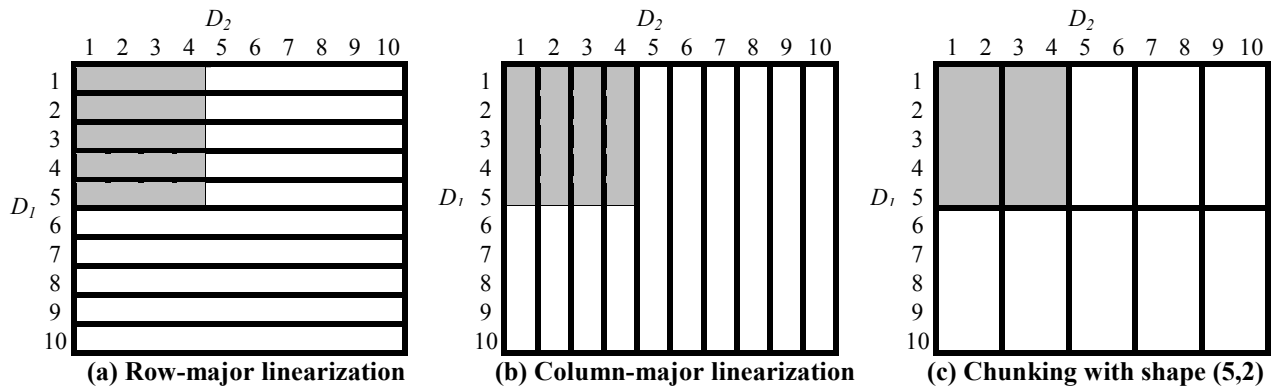


Figure 1: Nested traversals and Chunking of a 2-dimensional array

## 2. Background

As mentioned above, traditional linearization by any space filling curve does not take into account the query access pattern. So it is not quite fair to compare these approaches to the chunking approach which assumes the availability of access pattern information. We, nonetheless, need to make such a comparison to show the benefits of chunking. For this purpose we compare chunking to a very common space filling curve approach, namely, the nested traversal (sweep curve).

Consider an array of  $n$  dimensions:  $X[D_1, D_2 \dots D_n]$  where  $D_i$  is the length of the  $i^{\text{th}}$  dimension of the array. A linearization order based on any of the space filling curves, like the nested traversal approach, would choose one of the  $n!$  possible dimension orders and use it to linearize the array. For example, if  $n=2$ , there are two nested traversal alternatives: the *row-major* and the *column-major* orders.

Sarawagi and Stonebraker [2] introduced the basic chunking approach. The idea is to decompose the array into smaller sub-arrays, each having the same dimensionality as the original array. Each sub-array is called a *chunk*. The *chunk-size* is the total number of elements per chunk. The *chunk-shape* is the length of the chunk in the direction of each dimension. The optimization target is to find the optimal chunk size and the optimal chunk shape so that the average number of accessed disk blocks per query is minimized. The chunks themselves are then linearized by a nested traversal according to some dimension order. Hence, chunking does not cancel linearization; it only ‘defers’ it. Linearization is ultimately performed; but at a coarser level: chunks are linearized instead of array elements. The chunk size was chosen to be the size of the disk *I/O* block and the chunk shape was chosen such that it minimizes the average number of accessed disk blocks per query. It was shown by experiment that chunking outperforms the traditional nested traversal linearization. The following simple example illustrates and compares the nested traversal and the chunking approaches.

**Example:** Consider a 2-dimensional array of shape (10,10) and a range query  $Q$  that retrieves a rectangle of

shape (5,4). Assume that the array is to be stored on a device with an *I/O* block size of 10 array elements. Figure 1 illustrates the organization of this array using the nested traversal approaches and the chunking approach:

1- Nested Traversal: In figure 1(a) the array is linearized in row-major order: the query  $Q$  accesses 5 blocks (rows). In figure 1(b) the array is linearized in column-major order: the query  $Q$  accesses 4 blocks (columns).

2- Chunking: This approach suggests chunking the array into chunks each of which has a size of 10 (equal to the block size) and a shape of (5,2) as in figure 1(c). In this case, the query  $Q$  accesses only 2 blocks (chunks).

Note that several chunk shapes are possible; namely, (2,5), (5,2), (1,10), and (10,1). The shape (5,2) was chosen as optimal after trying all the shapes and picking the one that gives the best performance i.e. the least number of retrieved blocks. Note that the optimal chunk shape depends on the query shape; for example if  $Q$  were of shape (4,5) instead of (5,4) then the optimal chunk shape would have been (2,5) instead of (5,2). Hence, this approach assumes the availability of some information about the expected access pattern of queries. This information can be provided by the system users or by statistically sampling a set of real queries to determine the expected different query shapes and their relevant frequencies. A formal model of the access pattern is given in section 4.

## 3. Multilevel-Chunking

This section introduces the proposed multilevel chunking approach; the approach is mainly based on the physical and mechanical characteristics of hard disks. The ideas, however, can be generalized to other systems. The approach is based on two independent observations; the first observation is the hierarchical (multilevel) structure of hard disks: a hard disk is a collection of cylinders, a cylinder is a collection of tracks, a track is a collection of blocks, and a block is a collection of bytes. The second observation is that some access operations, like the cylinder seeking operation, are sequential operations. The suggested approach uses the idea of chunking together

with these two observations to improve the access performance of range queries on hard disks. These two observations and the ways they are used with the idea of chunking are explained in the following two subsections.

### 3.1. Hard Disk Hierarchy

The example in section 2 showed that the chunking approach can minimize the number of accessed disk blocks for a given query. We note that reducing the number of accessed blocks does not imply reducing the total I/O time because this number does not reflect the number of accessed tracks and cylinders. This suggests that in addition to reducing the number of accessed blocks, we also should reduce the number of accessed tracks and the number of accessed cylinders. This can be done by clustering the accessed blocks to fit in a small number of tracks and, similarly, clustering the accessed tracks to fit in a small number of cylinders. We suggest implementing these two clustering operations simply by applying the chunking approach at two levels above the first level which job is to cluster the array elements to fit in a small number of disk blocks.

So, the hard disk hierarchy suggests applying the chunking approach at 3 levels as follows:

- 1- *level-1*: Cluster the array elements in disk blocks.
- 2- *level-2*: Take the blocks outputted from *level-1* and cluster them in disk tracks.
- 3- *level-3*: Take the tracks outputted from *level-2* and cluster them in disk cylinders.

### 3.2. Sequential Access Operations

The approach proposed so far is a direct extension to the approach of level-1 chunking introduced in [2]. All what is done up till now is minimizing the number of accessed [blocks | tracks | cylinders] by applying the chunking process at three levels. In this subsection, we extend the chunking approach to a fourth level; the idea is, again, based on the mechanical features of hard disks.

The observation here is that the cylinder seek operation is sequential. This means that accessing a cylinder after another requires spanning all the cylinders in-between. Hence, after minimizing the number of accessed cylinders in level-3 chunking, we can get further improvement by clustering these accessed cylinders to make them physically close to each other. Unlike [2], which achieve this clustering by a nested traversal linearization of blocks, we suggest clustering the cylinders outputted from level-3 in 'groups of cylinders' by chunking. We call this: *level-4* chunking. Unlike the previous three levels, the target of this level is to minimize the number of spanned chunks and not the number of accessed chunks.

Note that some disk access operations are random rather than sequential. For example, switching from a track to another at the same cylinder is a random access operation as it is an electronic switch rather than a mechanical one. Also, the operation of switching from a block to another on the same track is performed randomly

in modern hard disks in which a track is buffered in random access storage once a block is accessed from that track. Finally, the operation of accessing array elements of the same block is random because the entire block is read atomically from the disk to the system main storage where all the access is random.

Only sequential access operations would benefit from clustering in the sense described by this observation. This is because the spatial locality of the accessed items achieved by clustering is not beneficial if the items, e.g. tracks on a certain cylinder, are randomly accessed.

Note that whatever the number of implemented chunking levels is (1, 2, 3, or 4), we still need to linearize the chunks outputted from the final chunking level. We call this: *Final Linearization*.

## 4. The Formal Model

This section presents a formal model for the problem and the suggested 4-levels chunking strategy.

### 4.1. The Access Pattern

The expected access pattern is the set of range queries expected to be applied to the array. As in [2], we model each user access request (range query) as an  $n$ -dimensional rectangle located somewhere within the array; and we group user accesses into collections of classes  $L_1...L_k$  such that each class  $L_i$  contains all rectangles of a specific size  $(A_{i1}...A_{in})$  located anywhere within the array. Each class has a weight or probability  $P_i$ , this weight represents the importance of that class. The sum of the class weights is 1.0. The access-pattern for an array is described by the set:  $\{(P_i, A_{i1}, A_{i2}, \dots, A_{in}) : 1 \leq i \leq k\}$ . Figure 2 illustrates an example of a 10x10 array; the three shaded rectangles, each accessed with probability  $\frac{1}{3}$ , represent access in two classes corresponding to the following access pattern:

$$\{(\frac{2}{3}, 3, 4), (\frac{1}{3}, 5, 3)\}$$

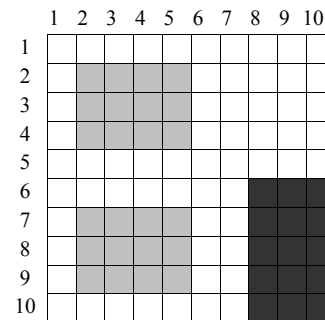


Figure 2: The Access Pattern Model

### 4.2. The Proposed Strategy

In this section, we give a formal description of the 4-level strategy suggested in section 3. The optimization problems at the first three levels are the same except that chunks of level-1 are disk blocks, chunks of level-2 are tracks and chunks of level-3 are cylinders. The problem at

level-4 is different from the first 3 levels as will be described below:

**1- At level-1:** Given The access pattern set:  $\{(P_i, A_{i1}, A_{i2}, \dots, A_{in}) : 1 \leq i \leq k\}$  and the chunk size  $C$  which is the number of elements per disk block, it is required to determine the optimal chunk shape  $(c_1, c_2, \dots, c_n)$  where  $c_i$  is the length of the  $i^{th}$  axis of the chunk. The optimal shape is defined as the shape that minimizes the cost function which is defined as “The expected number of accessed chunks (blocks) per query”.

As shown in [2], the total expected cost is:

$$\text{Total Expected Cost} = \sum_{i=1}^k \left( \prod_{j=1}^n \left\lceil \frac{A_{ij}}{c_j} \right\rceil \right) P_i .$$

To compute the optimal chunk shape, we simply list all the shapes of size  $C$ , i.e. those that satisfy the condition

$\prod_{i=1}^n c_i = C$  Then, for each shape, we evaluate the cost function and select the shape that minimizes the cost function. The selected shape is the chunk shape at level-1; i.e. the disk block shape in terms of array elements.

**2- At level-2:** First, we express the access pattern and the array shape in terms of the chunks outputted from level-1, i.e. the disk blocks. This is done by dividing each length in dimension  $i$  by the chunk length  $c_i$ . Then, we find the chunk shape at level-2; i.e. the track shape in terms of blocks, in the same way as level-1. Here, the chunk size  $C =$  the number of blocks per track.

**3- At level-3:** Similarly, we express the access pattern and the array shape in terms of tracks and get the chunk shape at level-3, i.e. the cylinder shape in terms of tracks. Here, the chunk size  $C =$  the number of tracks per cylinder i.e. the number of data surfaces in the disk.

**4- At level-4:** Similarly, we express the access pattern and the array in terms of cylinders. At this level, the optimization problem is different than the previous levels in two ways:

(1) The optimization variable is not only the chunk shape; but there are two more variables: (a) The chunk size  $C$ , i.e. the number of cylinders per group of cylinders. We don't have a closed formula for that, so we get  $C$  by trying a set of sizes and picking the one that performs best

(b) The optimal way to linearize the chunks outputted from this level (The final linearization). Specifying this way involves specifying both a space filling curve and a dimension order of the  $n!$  possible orders. For simplicity, we always use the nested traversal (sweep) curve. The remaining question is about the dimension order. [2] shows that given a chunk size and shape, it is possible to determine the optimal final linearization order, i.e. the one that minimizes the number of spanned cylinders, using a closed form solution instead of performing an exhaustive search.

(2) The cost function is the average number of spanned (instead of accessed) cylinders over all the query shapes of the access pattern. For a certain query shape, it can be shown [2] that the number of spanned cylinders is equal to

$$((z_1 - 1)(d_2 d_3 \dots d_n) + \dots + (z_{n-1})(d_n) + z_n) * C .$$

Where  $(z_1, z_2, \dots, z_n)$  is the query shape and  $(d_1, d_2, \dots, d_n)$  is the array shape; both in terms of the chunks of level-4, i.e. in terms of groups of cylinders.

Hence, the output of the optimization at this level is (1) chunk size (2) chunk shape, and (3) final linearization order.

## 5. Experimental Results

The source of the dataset that we used is the ocean model output from the General Circulation Model (GCM) simulations done at UCLA [2]. The array is a 5-dimensional one with a total size of 324.00 MB. We have assumed an access pattern conforming to the queries described in [2]

To examine the effect of the different levels of chunking, we measured the performance after implementing each level, so we have 4 cases. In all cases, the final linearization is performed as a nested traversal with the dimension order that minimizes the number of spanned cylinders. Figure 3 illustrates the performance after each chunking level. It shows that Multilevel Chunking improved the overall performance by reducing the average access time

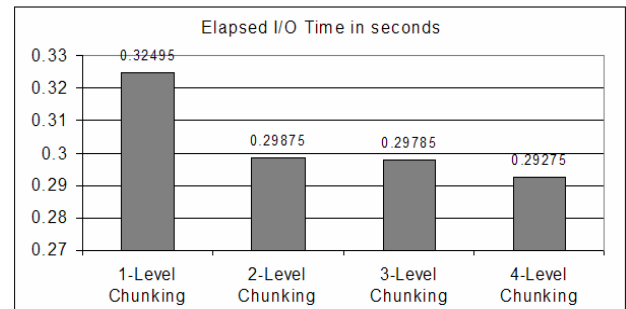


Figure 3 Experimental Results

## References

- [1] Volker Gaede, Oliver Günther: Multidimensional Access Methods. ACM Computing Surveys. Vol. 30, No.2, June 1998, 170-231
- [2] Sunita Sarawagi, Michael Stonebraker. Efficient Organization of Large Multidimensional Arrays. In *proceedings of the International Conference on Data Engineering ICDE*, 1994: 328-336