Extendible Multidimensional Array Based Storage Scheme for Efficient Management of High Dimensional Data

SK. MD. MASUDUL AHSAN
and
K. M. AZHARUL HASAN
Department of Computer Science and Engineering,
Khulna University of Engineering & Technology, Khulna 9203, Bangladesh.

Multidimensional arrays are good storage for managing large amount of data especially scientific and engineering applications. The Traditional Multidimensional Array (TMA) is also efficient in accessing the array elements by computing the addressing function. Thats why array based files are widely used. But TMA is not dynamically extendible during run time i.e the length of dimension and number of dimension is fixed for a TMA. We describe an extendible array file that is dynamically extendible during run time. If the length of dimension and number of dimension of a multidimensional array is large then the address space required for the array overflows quickly. The proposed array scheme handles the address space problem efficiently. The main idea of this scheme is to represent n dimensional array by a set of two dimensional extendible arrays. We evaluate our proposed scheme both analytically and experimentally for different array operations. Our experimental result shows that the proposed scheme outperforms the existing methods.

Keywords: Multidimensional Array, Extendible Array, Karnaugh Map, Range key Query, Dynamic Extension, MOLAP

1. INTRODUCTION

There are few classes of data structures which are as well understood or as extensively used as arrays. It is quite often for scientific, statistical and engineering applications to have computation on large multidimensional arrays for modeling and analyzing scientific phenomena (Seamons & Winslett, 1994; Sarawagi & Stonebraker, 1994). The strong need to handle large scale data efficiently has been promoting comprehensive research themes on organization or implementation schemes for multidimensional arrays.

The fast random accessing capability of multidimensional arrays is a fascinating characteristic that enables various statistical computations including aggregation to be performed efficiently (Li & Srivastava, 2002; Zhao et al., 1997). But this capability depends on the fact that the size of each dimension should be fixed so that a simple addressing function can be used to access an arbitrary element of the array. However, such kind of multidimensional arrays go through following two important problems:

—The size of the multidimensional array is not dynamically extendible (Hasan et al., 2009; 2005) when a new data value is added, size extension along the corresponding dimension is necessary and this implies reorganization of the entire array.

—One more problem with the multidimensional array is the contiguous address space requirement. To allocate memory, consecutive memory location is required for multidimensional array. But when the length of dimension and number of dimension of a multidimensional array is large then the address space overflows soon.

Authors' Email: SK. MD. Masudul Ahsan AHSAN (smmasudulahsan@yahoo.com)
and
K. M. AZHARUL HASAN (azhasan@gmail.com)

The introduction of extendible array (Hasan et al., 2009; 2005; Otoo & Merret, 1983; Ahsan & Hasan, 2011) solves the problem (i) above. An extendible array can be extended in any dimension without any repositioning of previously stored data (Hasan et al., 2005; Otoo & Merrett, 1983). Such advantage makes it possible for an extendible array to be applied into wide application area where required array size cannot be predicted before and/or can vary dynamically during operating time of the system. In this paper, we are going to propose a basic extendible data structure that will solve the problem (i) above. To solve the second problem we have segmented the subarrays in such a way that the subarrays are always 2 dimensional that solves problem (ii) in our model.

An n dimensional Array $A[l_1, l_2, l_3, \ldots, l_n]$ is an association between n-tuples of integer indices $\langle j_1, j_2, \ldots, j_n \rangle$ and the elements of a set of E such that, to each n-tuples given by the ranges $0 \leqslant j_1 \leqslant l_1, 0 \leqslant j_2 \leqslant l_2, \ldots, 0 \leqslant j_n < l_n$ there corresponds an element of E. The domain from which the elements are chosen is immaterial and we make the assumption that k bytes of storage area are needed to each n-tuples. The set of contiguous memory locations into which the array maps is denoted by $A[0 : D]$, where $D = ((\Pi_{i=1 \ldots nl_i}) - 1)$ and the size of the array is denoted by $S = D \times k$. If number of dimension n and the length of each dimension $l_i (1 \leqslant i \leqslant n)$ increase the total address space or array size S becomes very large that causes the existing data types to overflow even for 64 bit machines. Hence it is impossible to memory for allocate such a large size multidimensional array.

An extendible array, however, does not store an individual array; rather, it stores an array and all its potential extensions. In this paper, we propose a new scheme called Extendible Karnaugh Array (EKA) (Ahsan & Hasan, 2011). The main idea of our proposed EKA is to represent multidimensional array by a set of two dimensional extendible arrays. To evaluate our proposed scheme we have implemented and compared the results with TMA. The experimental result shows that EKA outperforms the TMA in various cases. The scheme is fully general to be applied in different applications such as implementation of multidimensional database systems (Hasan et al., 2005; Pedersen & Jensen, 2001), or data warehouse systems (Li & Srivastava, 2002; Hasan et al., 2007). It can also be applied to design extendible and flexible database (Markus & Gabriele, 2005; Roland & Bayer, 2005) and parallel database design (Hasan et al., 2006; Chen et al., 2006)

2. RELATED WORK

There are some well known existing multidimensional array systems to represent multidimensional data such as Traditional Multidimensional Array (TMA) (Seamons & Winslett, 1994; Sarawagi & Stonebraker,1994), Extended Karnaugh Map Representation (EKMR)(Chun et al., 2002; 2003), Extendible multidimensional Array (Otoo & Meritt, 1983; Rotem & Zhao, 1996) . TMA is a good storage for storing multidimensional data set. The TMA represent n dimensional data by an array cell in an n dimensional array. The TMA is modeled in the positive orthant of n-dimensional space where array positions lay on the lattice points. To extend the TMA dynamically, it is necessary to reorganize the entire array which causes massive cost?(Hasan et al., 2007). One more problem with the TMA is that if the length of dimension and number of dimension increases then the address space requirement overflows soon. Extended Karnaugh Map Representation (EKMR) represent n dimensional data by a set of two dimensional arrays(Li & Srivastava, 2002). But EKMR is not dynamically extendible and the length of dimension as well as number of dimensions is same as TMA, hence it also overflows like TMA.

The Extendible Array (Otoo & Meritt, 1983; Rotem, & Zhao, 1996) has the property of extendibility and the indices of the respective dimensions can be arbitrarily extended without reorganizing previously allocated elements. If the extendible array is n dimensional then the subarray is $n-1$ dimensional. Hence it will overflow soon for address space because the subarrays are $n-1$ dimensional although it can be extended dynamically. The extendible array is employed by Otoo & Meritt (1983) to extend the array and it only treats an organization scheme of the history tables. The subarrays are n dimensional hence it will be difficult to apply in actual implementation when address space overflow is concerned.

An extendible array model proposed by Otoo & Rotem (2006) where there is record for each dimension called axis vector. Each element of the vector stores necessary information like the auxiliary tables in our EKA scheme to retrieve an element correctly. In this approach the sequence of the two consecutive extensions along the same dimension, although occurring at two different instances, is considered as an uninterrupted extension of that dimension and handled by only one expansion record entry in the axial-vector. Therefore number of element in an axial vector is always less than or equal to the number of indices of the corresponding dimension. If round robin expansion occurs, then it is similar to extendible array Otoo & Meritt (1983). Zaho et al. (1997) and Rotem et al. (2007) present the chunking of multidimensional arrays. In this scheme the large multidimensional arrays are broken into chunks for storage and processing. All the chunks are n dimensional with smaller length than the original array. But the dynamic extension is not possible. Kumakiri et al. (2006) proposed an array system based on extendible array proposed by Otoo & Meritt (1983) that can insert a row in the middle of the dimension. All the array models mentioned in this Section do not handle the address space overflow. Many of them have a concept of subarray which is $n-1$ dimensional if the array having n dimensions. Maximum value of coefficient vector is fairly large even for $n-1$ dimensional subarray, and quickly overflows. On the other hand, the proposed EKA is a set of two dimensional arrays; therefore maximum value of coefficient vector is relatively small even for large number of dimensions. And hence, delays the overflow.

3. THE EXTENDIBLE KARNAUGH REPRESENTATION OF ARRAY

In this section we are going to propose an extendible array model namely Extendible Karnaugh Array (EKA). The idea of EKA is based on Karnaugh Map (K-map) (Mano, 2005). The K-map is used for minimizing Boolean expressions usually aided by mapping values for all possible combinations. Fig. 1 (a) shows a 4 variable K-map to represent possible 24 combinations of a Boolean function. The array representation of K-map for 4 variable Boolean function is shown in Fig. 1(b). The length of each of the dimensions is 2 for both Fig. 1(a) and (b). This is because the Boolean variables are binary that causes the length to be 2.

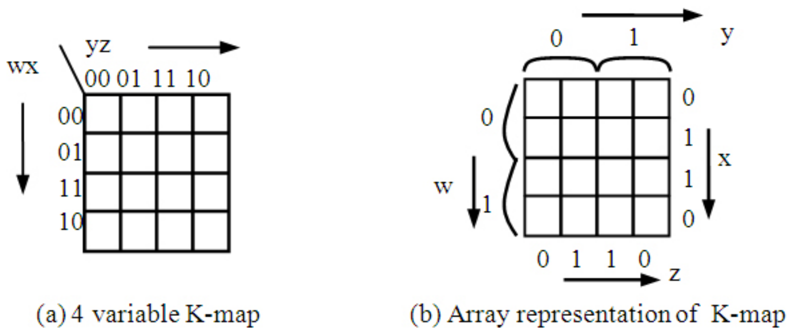


Fig. 1. Realization of Boolean function using K-map.

Definition 1 (Adjacent Dimension): The dimensions (or index variables) that are placed together in the Boolean function representation of K-map are termed as adjacent dimensions (written as $adj(i) = j$). The dimensions (w, x) are the adjacent dimensions in Fig. 1(a) and (b) i.e. $adj(w) = x$ or $adj(x) = w$.

3.1 The Extension Realization of the EKA Scheme

EKA is an array system that is the combination of subarrays. It has three types of auxiliary tables namely history table, coefficient table and address table. For each dimension these tables exist. These tables help the elements of the EKA to be accessed very fast. The subarrays are further divided into segments. The number of segments determines the number of entries in the address table and is calculated from the length of adjacent dimension. The subarrays are always 2 dimensional for an n dimensional EKA, $EKA(n)$.

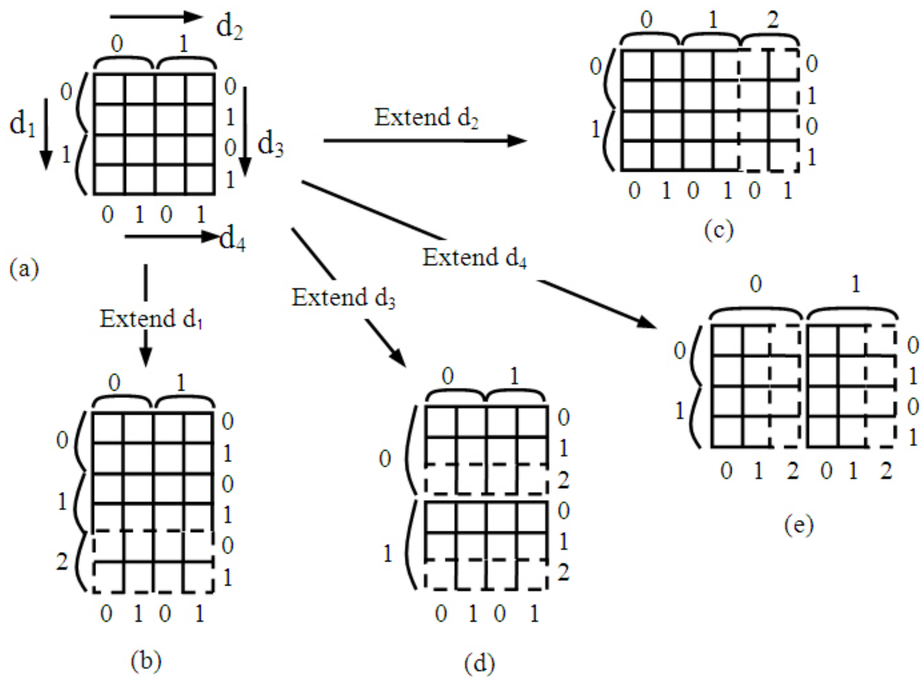


Fig. 2. Logical extension of 4-dimensional EKA.

Fig. 2 presents a 4-dimensional array with a logical view of the array after extending in a dimension. During each extension the history table is populated with a strictly monotonically increasing number, i.e. it counts the construction history. Address table stores the first addresses of each segments of the subarray. In an n dimensional conventional fixed size array of size $[l_n, l_{n-1}, \ldots, l_1]$ is allocated on memory using an addressing function like this:

$$f(x_n, x_{n-1}, \ldots, x_2, x_1) = l_1 l_2 \ldots l_{n-1} x_n + l_1 l_2 \ldots l_{n-2} x_{n-1} + \cdots + l_1 x_2 + x_1$$

Here, $l_1 l_2 \ldots l_{n-1}, l_1 l_2 \ldots l_{n-2}, \ldots, l_1 l_2, l_1$ is called as *coefficient vector*. Coefficient table holds the coefficient vector of each segment. Since segments are 2 dimensional, in our model, the coefficient vector is simply l_1.

Fig. 3 shows the dynamic extension of an $EKA(4)$. Fig. 3(a) illustrates the initial setup of the scheme. The history counter is zero and the history tables contain one entry namely 0. The address tables contain first address. Each of the coefficient table contains value 1 since length of each dimension is 1. During the extension of d_1 and d_3 dimension size of the segment is $l_2 \times l_4$ which is a two dimensional array, and so coefficient vector is one dimensional. Hence, for our example we use l_2 as coefficient vector for d_1 and d_3 dimensions. Similarly, l_3 is used as coefficient vector for d_2 and d_4 dimension and coefficient table is maintained. When an extension along d_2 direction is done as shown in Fig.3(b), the history counter is increased by 1. The value of history counter is stored in the history table H_{d2}. The subarray size $[l_1, l_3, l_4]$ is calculated and dynamically allocated; the values of first address are stored in address table A_{d2}; since l_3 is 1 C_{d2} stores this value.

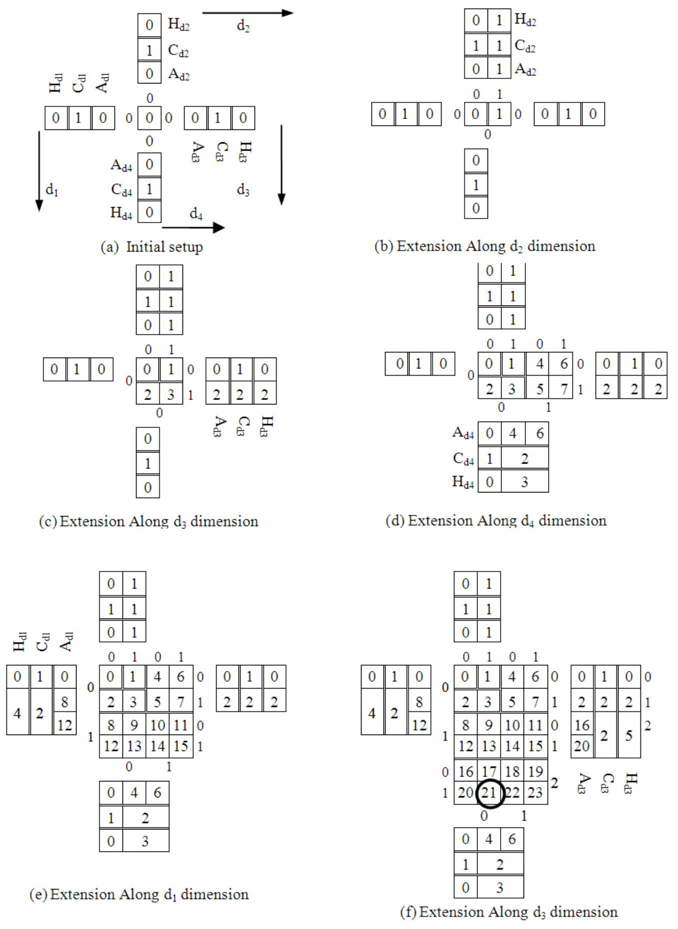

(a) Initial setup
(b) Extension Along d₂ dimension
(c) Extension Along d₃ dimension
(d) Extension Along d₄ dimension
(e) Extension Along d₁ dimension
(f) Extension Along d₃ dimension

Fig. 3. Extension realization of EKA(4).

Fig. 3(c) shows an extension along d_3 direction. Here the history counter incremented by 1 and this is stored in history table H_{d3}. The size of the subarray $[l_1, l_2, l_4]$ is calculated and the first address for this subarray is stored in the address table A_{d3}. In Fig. 3(d) an extension along direction d_4 is done. As a result of the extension history counter becomes 3 and subarray size becomes 4. Here, H_{d4} memorizes the value of 3. Similarly, the extension along direction d_1 is shown Fig. 3(e) and finally Fig. 3(f) shows one more extension along direction d_3.

3.2 Generalization to Higher Dimensions

The EKA scheme can be generalized to n dimensions using a set of $EKA(4)$s. Fig. 4(a) shows the logical structure and Fig. 4(b) shows the physical implementation of a $EKA(5)$ where the length of 5th dimension d_5 is 2. Fig. 5 shows an $EKA(6)$ represented by a set of $EKA(4)$ in two level. If the current length of 6th dimension (d_6) is 2 and 5th dimension (d_5) is 3 respectively, then the $EKA(6)$ is represented by the two level tree like structure as shown in Fig. 5. Each higher dimensions (d_5 and d_6) are represented as one dimensional array of pointers that points to the next lower dimension and each cell of d_5 points to each of the $EKA(4)$. So each $EKA(4)$ can be accessed simply by using the subscripts of higher dimensions. For the case of $EKA(n)$, similar hierarchical structure will be needed. The set of $EKA(4)$s stores the actual data values and the hierarchical arrays are indexes to $EKA(4)$s and used to locate the appropriate $EKA(4)$. Hence the EKA(n) is a set of $EKA(4)$s and a set of pointers are used for indexing purpose only. At this stage (Fig. 5), if dimension d1 (or d_2, d_3, d_4) is extended dynamically, all the $EKA(4)$s will be extended along that dimension and the auxiliary tables are maintained. And if there is an extension in number of dimension i.e. addition of a new dimension (d_7), we will simply add a pointer array as a root of the tree and set the appropriate next level links and store the actual data values in the $EKA(4)$s.

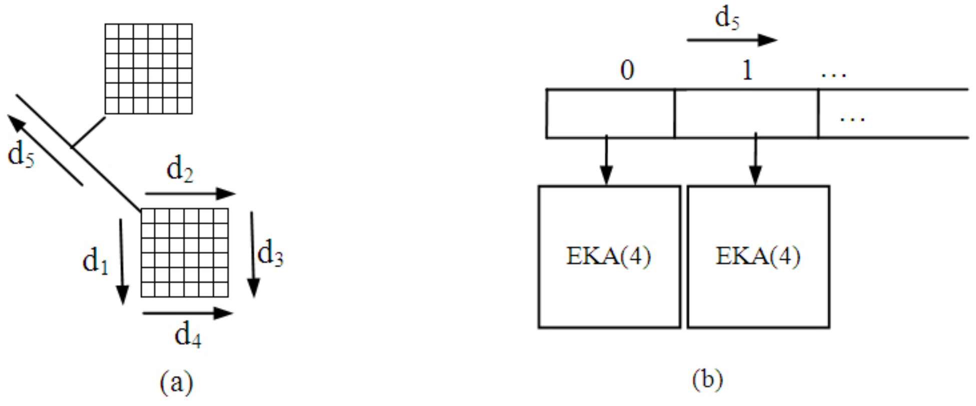


Fig. 4. Realization of 5-dimensional EKA.

4. OPERATIONS ON EKA

4.1 Retrieval

4.1.1 *Retrieval on EKA(4).* Let the value to be retrieved is indicated by the subscript $\langle x_1, x_2, x_3, x_4 \rangle$. The maximum history value among the subscripts
$h_{max} = max(H_{d_1}[x_1], H_{d2}[x_2], H_{d3}[x_3], H_{d4}[x_4])$ and the dimension (say d_1) that corresponds to history value h_{max} is determined. h_max is the subarray that contains our desired element. This is because; the subarray that has the maximum history value was constructed at last (since the history values are linear numbers). Hence the desired element remains in the subarray having history value h_{max}. Now the first address and offset from the first address is to be found out. The adjacent dimension $adj(d_1)$ (say d_3) and its subscript x_3 is found. The first address is found from $H_{d1}[x_1]$. $A_{d_1}[x_3]$. The offset from the first address is computed using the addressing function (described in Section 3.1); the coefficient vectors are stored in C_{d1}. Then adding the offset with the first address, the desired array cell for the given subscript can be found.

Example 1: Let four subscripts $\langle 1, 0, 2, 1 \rangle$ from Fig. 3(f)
$h_{max} = max(H_{d1}[1], H_{d2}[0], H_{d3}[2], H_{d4}[1]) = max(4, 0, 5, 3) = 5$, and dimension corresponding to h_{max} is d_3 whose subscript is 2 and $adj(d_3) = d_1$ and $x_1 = 1$. So the first address is in $A_{d3}[2][1] = 20$, and offset is calculated using the coefficient vector stored in coefficient table C_{d3} which is 2. Here offset $= C_{d3}[2] \times x_4 + x_2 = 2 \times 1 + 0 = 2$. Finally adding the first address with the offset the desired location 20 + 2 = 22 is found (encircled in Fig. 3(f)).

4.1.2 *Retrieval on higher dimensional EKA(n), $n > 4$.* Let the value to be retrieved is indicated by the subscript $(x_n, x_{n-1}, \ldots, x_2, x_1)$. Each of the higher dimensions ($n > 4$) are the set one dimensional pointer arrays that points to next lower dimensions. Hence using the subscripts $x_k(d_k > 4)$ the pointer arrays are searched to locate the lower dimensions (see Fig. 5). If $n \leqslant 4$, then using the above computation technique the location in $EKA(4)$ can be found.

4.2 Range Key Retrieval

A range key query (Bertin & Kim, 1989) has a single predicate of the form *(column subscript <value)* or *(column subscript <value)* or *(column subscript between* $value_1 and value_2$*)*. Fig. 6 (which is obtained by extending the Fig. 3(f) in d_2 dimension) shows the candidate range (bold dotted line) of a range key query for a $EKA(4)$. Assume that the candidate range of the subscripts of the corresponding dimension d1 has NRQ subscripts. Let the specified range involve in the known column has subscripts $x_{k1}, x_{k2}, \ldots, x_{kNRQ}$ of dimension d_k. Let $h_1, h_2, \ldots, h_{NRQ}$ be the history values that correspond to the subscripts and the minimum of them is hmin.

Definition 2 (Major and Minor subarray): All the elements of the subarrays corresponding to the history values $h_1, h_2, \ldots, h_{NRQ}$ are candidate for retrieval (see Fig.6) and are called Major subarray. The subarrays that have history values greater than h_min and belong to the adjacent dimension $adj(d_k)$ are called Minor subarray. One or more segments of the minor subarrays are candidate for a range query. The candidate subarrays are those which are sufficient to be searched and these subarrays have history values greater than hmin. The major and minor subarrays are candidate subarrays and rest of the candidate subarrays do not belong to the known dimension dk and have history values greater than h_{min}.

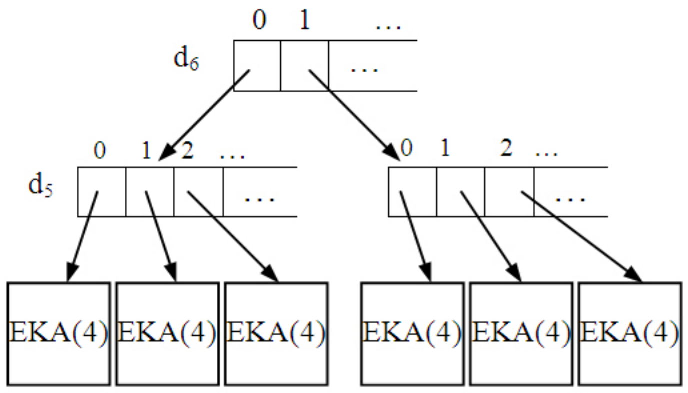


Fig. 5. Realization of 6-dimensional EKA

Example 2: In Fig. 6, the subarray having history value 4 is the major subarray and the subarray having history value 5 is the minor subarray. Hence all the elements of subarray 4 are candidate for retrieval and one segment of subarray 5 are candidate for retrieval. The remaining candidate subarrays have history value greater than 4 (see definition 2) and the elements inside the subarray are found by calculating the offsets and adding the first address as described in Section 4.1

5. THEORETICAL ANALYSES

In this section, we model the processes of retrievals and extensions for multidimensional array under two different implementation strategies namely Traditional Multidimensional Arrays (TMA) and our proposed Extendible Karnaugh Arrays (EKA). The TMA reorganizes the array whenever there is an extension to it. That is, the whole array will be relinearized on disk to accommodate the new data due to the extension of length of dimension. The second strategy extends the initial array with segment of subarrays containing the new data as described in Section 3. In this Section, we show that the EKA strategy can reduce the cost of array extensions significantly. We will derive the cost functions for both extensions and retrievals in the following. All the array schemes are assumed to be stored in secondary storage and performed the operations.

5.1 Parameters

The cost functions are represented as the number of array cells required to access. The parameters that are assumed are described in Table 1. All the lengths and sizes are in bytes. Some parameters are provided as input while others are derived from input parameters.

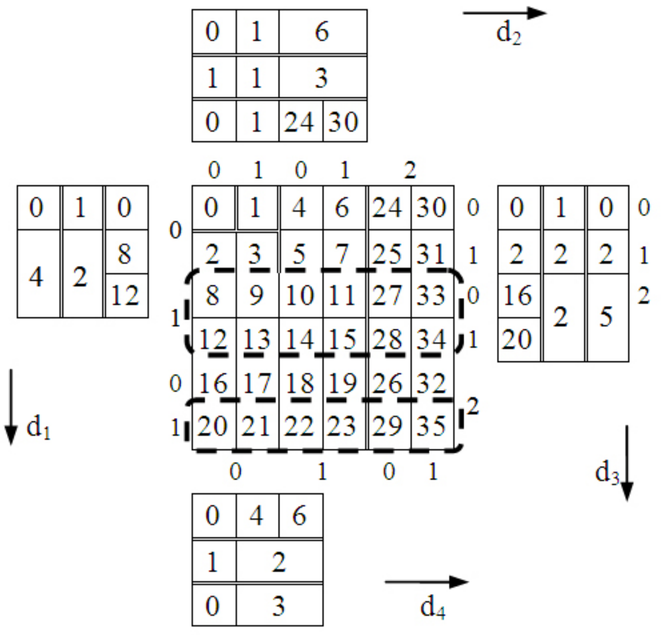


Fig. 6. Range query on EKA(4).

Assumptions: To simplify the cost model we make a number of assumptions.

—The length of dimension extends in round robin manner for both TMA and EKA.

—The length of each dimension is equal. We denote the length of dimension after ith extension as l_i.

—All the basic CPU operations are executed in constant time.

5.2 Retrieval Cost

In TMA, the array is linearized in a single data stream using the addressing function described in Section 3 and all offset values of the array elements are consecutive. Hence the range of candidate offset values for a query can be determined uniquely. But for EKA, the same data stream is distributed over different subarrays (See Fig. 6).

5.2.1 *Retrieval Cost for* TMA. The retrieval on TMA is dependent on the known dimension of an array. We use the term known dimension (or known subscript) to indicate the specified dimension of the query operation. In an n dimensional TMA, if the query is along dimension n (i.e. subscript x_n is known) then all the candidate offsets are consecutive and the volume of the range of the query is l_i^{n-1}. This is explained with an example in the following. The addressing function of a 4 dimensional array with $l_i = l$ is $f(x_4, x_3, x_2, x_1) = l^3x_4 + l^2x_3 + lx_2 + x_1$

If $l = 6$ and x_4 is known (say, $x_4 = 0$, and $x_j = 0, \ldots, l-1$) then the candidate offset values in the query are consecutive in the range 0 to 215 (total 216 offsets) out of 1296 offsets which is l^3 (i.e. 63). If x_1 is known (say, $x_1 = 0$) then the candidate offset values in the query are in the range 0 to 1290 (total 1291 offsets) out of 1296 offsets. Hence the volume of the candidate range of target elements are determined by $l^4 - (l-1)$. If the subscript x_2 is known then the volume of the candidate range of offsets is $l^4 - l(l-1)$. In general, if the subscript $x_k (1 \geqslant k \geqslant n)$ is known then the volume of the target elements are determined by $l^n - l^k - 1(l-1)$. For the range key query in the range of known subscripts NRQ along the dimension k, the volume of the target elements are determined by $NRQ \times (l^n - l^k - 1(l-1))$.

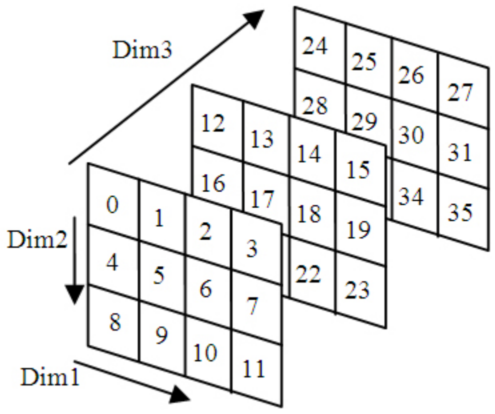


Fig. 7. A 3 dimensional TMA realized in row major order.

Example 3: Consider a 3 dimensional array of size $3 \times 3 \times 4$ stored as row major order shown in Fig.7. let the known subscript is $x_1 = 0$, then, the candidate values for the query can be represented as shown in Fig 8(a). Or if $x_1 = 1$ the candidate values would be as in Fig. 8(b), and so on, i.e. each of the candidate values are totally discrete and spread over the entire range. If the known subscript is $x_2 = 1$ then, the candidate values can be shown as in Fig. 8(c). Here some of the candidate values are grouped together. If the known dimension is 3, and $x_3 = 0$ then, the candidate values can be represented as shown in Fig. 8(d). Here many of the candidate values are contiguous in nature and a single read is enough to collect them.

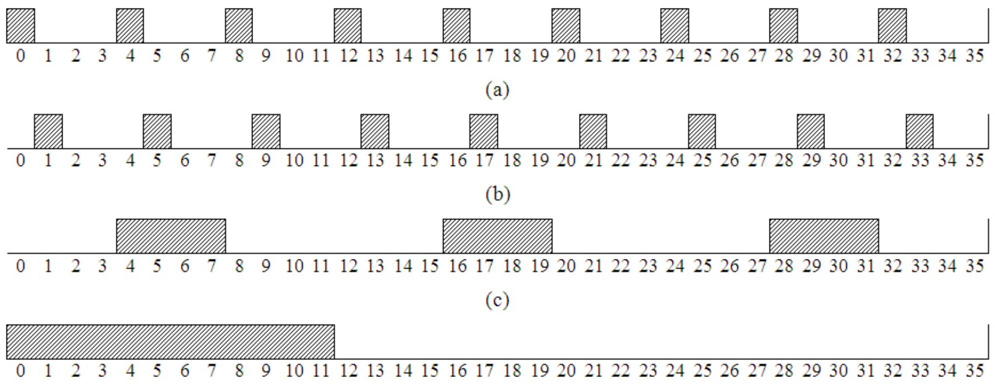


Fig. 8. Retrieval candidates of a 3 dimensional TMA.

5.2.2 *Retrieval Cost for EKA.* In EKA scheme, the target elements are distributed in different subarrays and the subarrays are divided into segments (see Fig. 6). So for retrieval operations, EKA will take more CPU operation to be performed for accessing different streams in secondary memory. On the other hand each of the segments of the subarray is two dimensional and candidate and non candidate items can be separated in EKA. And thus retrieval cost will be lower. As the segments are 2 dimensional then the maximum volume of the target elements for a query in a segment is determined by $NRQ \times (l^2 - (l-1))$. If the number of segment is s then the maximum volume of the target elements are determined by $s \times NRQ \times (l^2 - l(l-1))$; where s depends on the size of the subarray.

5.3 Extension Cost

Fig. 9(a) shows a 2D TMA of size 3×4. Let the italicized values represent the location of each cell after linearization. From Fig. 9(a), the location of cell $\langle 2, 1 \rangle$ is 9 and its value is 76. Now let we extend the array one unit in d_2 dimension. Fig. 9(b) shows the array after extension and linearization, where location of cell $\langle 2, 1 \rangle$ is now 1_1. That is if we would simply append the extension subarray at the end, and read from location 9 for subscript $\langle 2, 1 \rangle$ we will get wrong value (82). After extension, location is changed due to the change in the parameters of addressing function. So for exact retrieval, during extension we first need to read the previously allocated data, reorganize them and then write the array along with extension subarray.

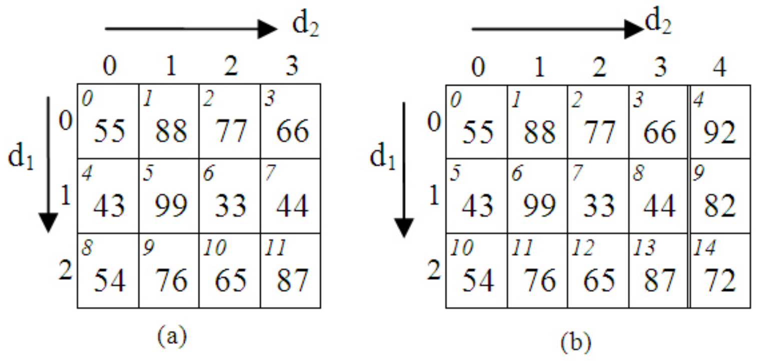


Fig. 9. A 2 dimensional TMA and its extension.

5.3.1 *Extension Cost for TMA.* Let a $TMA(n)$, with length of each dimension $l_i = l$. Therefore initial volume of the array, $V = l_1 \times l_2 \times l_3 \times \cdots \times l_n = l^n$. For extending TMA, it requires to reorganize the array and rewrite both existing and new data elements. The existing elements of the initial array need to be faced and recalculate the new offsets due to the extension for TMA.

Hence the cost of facing (FC) the existing array elements becomes, $FC_{TMA} = l^n$ (Assumption (ii)). If TMA is extended by λ unit then a new TMA of length $l + \lambda$ is to be reallocated, hence reallocation cost, $RC_{TMA} = (l_1 + \lambda) \times (l_2 + \lambda) \times \cdots \times (l_n + \lambda) = (l + \lambda)^n$

So, Total extension cost for $TMA(n)$

$$EC_{\lambda}^{TMA(n)} = FC_{TMA} + RC_{TMA} = l^n + (l+\lambda)^n = l^n + \textstyle\sum_{i=0}^{n} {}^nC_i l^{n-i} \lambda^i$$

$$= l^n + {}^nC_0 l^n + \sum_{i=1}^{n} {}^nC_i l^{n-i} \lambda^i = 2l^n + \sum_{i=1}^{n} {}^nC_i l^{n-i} \lambda^i \qquad (1)$$

5.3.2 *Extension Cost for EKA.* First we consider an $EKA(4)$ with $l_i = l$. Therefore, initial volume of the array before extension $V = l_1 \times l_2 \times l_3 \times l_4 = l^4$.
If we extend one unit along each dimension d_i, the size of extension subarray SE_i are
$SE_1 = l_2 \times l_3 \times l_4 = l^3$, and due to this extension $l_1 = l + 1$
$SE_2 = l_1 \times l_3 \times l_4 = (l + 1) \times l^2$, and $l_2 = l + 1 [\because l_1 = l + 1]$
$SE_3 = l_1 \times l_2 \times l_4 = (l + 1)^2 \times l$, and $l_3 = l + 1 [\because l_1 = l + 1, l_2 = l + 1]$
$SE_4 = l_1 \times l_2 \times l_3 = (l + 1)^3$, and $l_4 = l + 1 [\because l_1 = l + 1, l_2 = l + 1, l_3 = l + 1]$

Now in general, extending a λ unit along dimension d_i, the size of extension SE_i can be written as
$SE_1 = \lambda \times l_2 \times l_3 \times l_4 = \lambda l^3$, and for this extension $l_1 = l + \lambda$
$SE_2 = \lambda \times l_1 \times l_3 \times l_4 = \lambda(l + \lambda)l^2$, due to extension $l_2 = l + \lambda$
$SE_3 = \lambda \times l_1 \times l_2 \times l_4 = \lambda(l + \lambda)^2 l$, and $l_3 = l + \lambda$
$SE_4 = \lambda \times l_1 \times l_2 \times l_3 = \lambda(l + \lambda)^3$, and then $l_4 = l + \lambda$

Therefore, Total Extension Cost for $EKA(4)$, having λ unit extension in each dimension, becomes

$$EC_{\lambda}^{EKA(4)} = SE_1 + SE_2 + SE_3 + SE_4 = \lambda \sum_{i=0}^{k} l^{k-i}(l + \lambda)^i, \text{where } k = 3$$

Similarly for $EKA(n)$, Total Extension Cost, for λ unit extension in each dimension, can be written as

$$EC_{\lambda}^{EKA(n)} = SE_1 + SE_2 + \cdots + SE_{n-1} + SE_n = \lambda \sum_{i=0}^{k} l^{k-i}(l + \lambda)^i, \text{Where } k = n - 1 \ldots \quad (2)$$

If we expand the summation, then

$$\begin{aligned}\sum_{i=0}^{k} l^{k-i}(l + \lambda)^i = {} & l^k(l + \lambda)^0 + l^{k-1}(l + \lambda)^1 + l^{k-2}(l + \lambda)^2 + \cdots + l^1(l + \lambda)^{k-1} + l^0(l + \lambda)^k \\ = {} & l^k + \\ & l^{k-1}({}^1C_0 l +^1 C_1 \lambda) + \\ & l^{k-2}({}^2C_0 l^2 +^2 C_1 \lambda l +^2 C_2 \lambda^2) + \\ & l^{k-3}({}^3C_0 l^3 +^3 C_1 \lambda l^2 +^3 C_2 \lambda^2 l +^3 C_3 \lambda^3) + \\ & \vdots \\ & + l^0({}^kC_0 l^k +^k C_1 \lambda l^{k-1} + \cdots +^k C_{k-1} \lambda^{k-1} l +^k C_k \lambda^k)\end{aligned}$$

After multiplying and collecting the coefficients of $l^p, p = 0, 1, \ldots, k$, we get

$$\sum_{i=0}^{k} l^{k-i}(l+\lambda)^i = l^k \sum_{i=0}^{k} {}^iC_0 + l^{k-1}\lambda \sum_{i=1}^{k} {}^iC_1 + \cdots + \lambda^k \sum_{i=k}^{k} {}^iC_k$$

$$= {}^{k+1}C_1 l^k + {}^{k+1}C_2 l^{k-1}\lambda + \cdots + {}^{k+1}C_k l \lambda^{k-1} + {}^{k+1}C_{k+1}\lambda^k \qquad \left[\because \sum_{j=0}^{p} {}^jC_r = {}^{p+1}C_{r+1}\right]$$

$$= \sum_{i=1}^{n} {}^nC_i l^{n-i}\lambda^{i-1}, \text{Where } n = k+1$$

Putting the above value in eq. (2), we get
$EC_\lambda^{EKA(n)} = \lambda \sum_{i=0}^{k} l^{k-i}(1+\lambda)^i$, Where $k = n-1$,

$$= \lambda \sum_{i=1}^{n} {}^nC_i l^{n-i}\lambda^{i-1} = \sum_{i=1}^{n} {}^nC_i l^{n-i}\lambda^{i} \tag{3}$$

Extension Gain: The difference of extension cost between the TMA and EKA strategies is referred to as *Extension Gain* (EG), $EGn\lambda = EC_\lambda^{TMA(n)} - EC_\lambda^{EKA(n)} = eq.(1) - eq.(3) = 2l^n$. Hence we conclude that the extension gain of $EKA(n)$ over $TMA(n)$ is twice of initial volume of TMA. And EG is independent of the length of extension λ.

Table 2. Assumed parameters for constructed prototypes

n	λ	$max(l_i)$	Initial $V = l^n$
4	10	100	$(30)^4$
5	5	45	$(20)^5$
6	2	22	$(10)^6$

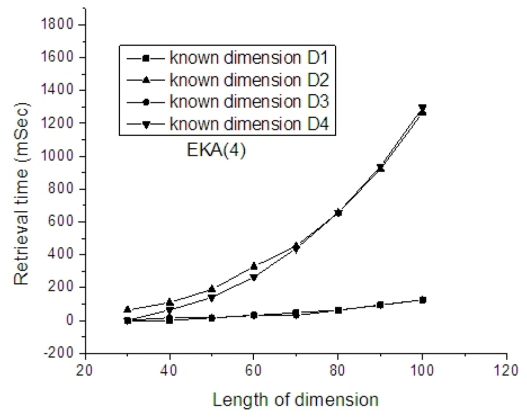
6. PERFORMANCE RESULTS

We have constructed the TMA and EKA systems having the parameter values shown in Table 2 placing the TMA and EKA in secondary storage. The auxiliary tables of EKA are placed in main memory since the sizes of the auxiliary tables are negligible comparing to the main array. All the tests are run on a machine (Dell Optiplex 380) of 2.93 GHz processor and 2 GB of main memory having disk page size 4KB. We will show that the overall retrieval time has advantages for EKA than TMA. We also show that we can extend the length of dimension of a multidimensional array effectively if implemented using EKA.

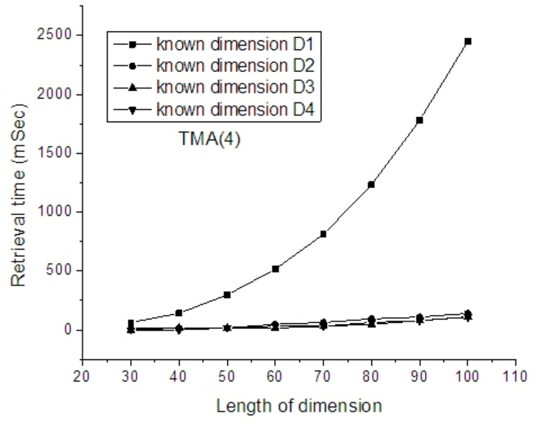
6.1 Retrieval Cost

Fig. 10 shows the retrieval performance for range key query of TMA and EKA for the parameter values shown in Table 2. In Fig. 10(a) the retrieval performance for $EKA(4)$ for different known dimension is shown. It shows that, the retrieval time is higher when x_2 and x_4 are known. The retrieval time is lower when the known x_1 and x_3 is known. This is because the segments of the subarrays of $EKA(4)$ are two dimensional hence the element inside the subarrays can be organized as row major order or column major order. If the elements are organized in one order (say row major) then it is searched in column order; the target elements for the query are not consecutively organized. Therefore that known subscript takes longer time. Hence two known subscripts will take higher time than other two known subscripts.

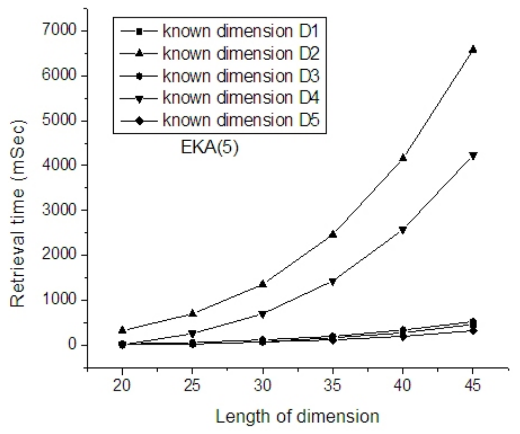
When the number of dimension n increases then it (Fig. 10(c) and 10(g)) shows that retrieval from EKA takes higher time the known subscripts of only two values. Fig. 10(b) shows the retrieval time for TMA for $n = 4$. It shows that d_1 dimension takes higher time than other known dimensions which proves the theory. When $n = 5$ or 6 for TMA, the same situation i.e. for one known subscript TMA takes higher time than others as shown in Fig. 10(d) and 10(h). Fig. 10(e) and 10(f) shows the average retrieval cost for EKA and TMA for $n = 4$ and 5. It can be concluded that, on average, the retrieval performance for EKA is better and there is no retrieval penalty for EKA over TMA up to $n = 6$. From a closer look to the absolute retrieval times, we will also find that EKA needs less time for some known dimension and gives fairness rather than TMA.



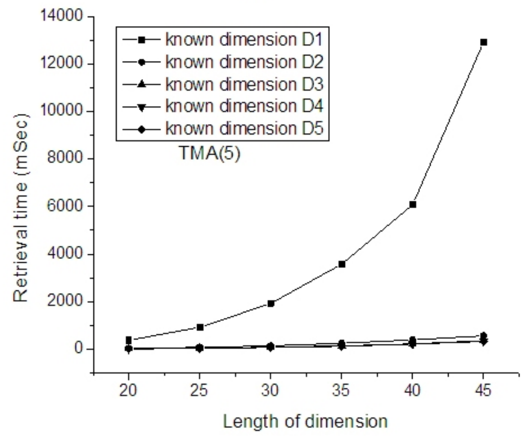
(a)



(b)



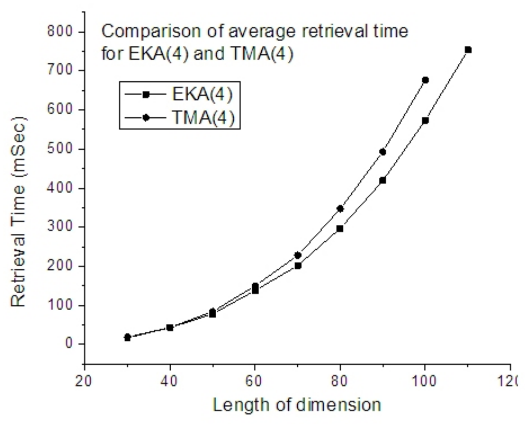
(c)



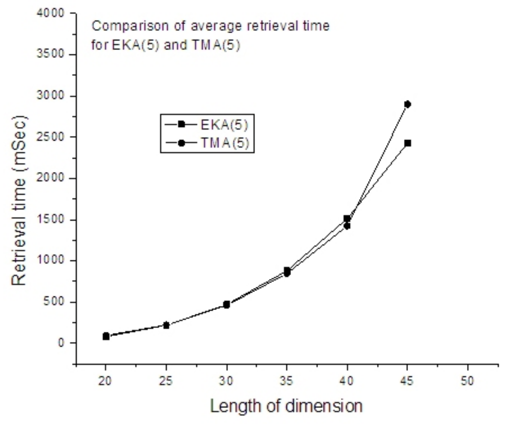
(d)

6.2 Retrieval Cost

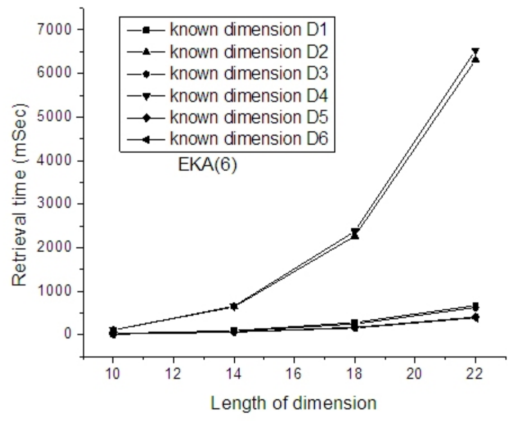
Fig. 11 shows the extension cost for TMA and EKA for the parameter values shown in Table 2. The extension gain is shown in Fig. 11(d).). The TMA needs to reorganize the entire array whenever there is an extension made on it. That is, the whole array will be relinearized on disk to accommodate the new data due to the extension of length of dimension. For this process, the TMA scheme needs to face the existing elements then reorganize for the extension. On the other hand, the EKA extends the initial array with segment of subarrays containing the new data as



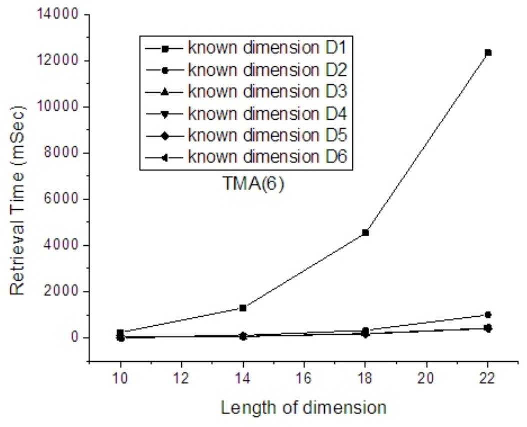
(e)



(f)



(g)



(h)

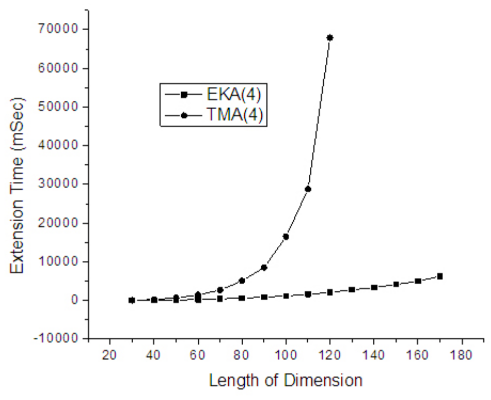
Fig. 10. Retrieval cost analysis for EKA and TMA.

described in Section 3. That is, the EKA strategy just append the extended subarray at the end, hence it can reduce the cost of array extensions significantly.

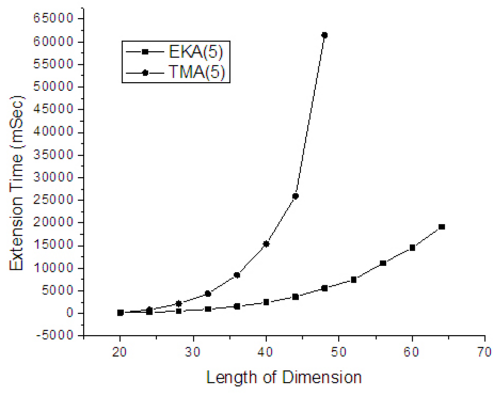
From theoretical analysis we can find that the extension cost is an exponential function both for TMA and EKA. For example the extension cost for TMA is approximately $O(l^n)$ whereas for EKA this value approximates to $O(l^{n-1})$. Extension times in Fig. 11(a), 11(b), and 11(c) resembles this phenomenon and hence we validate our cost model. The extension cost as well as extension gain depends on the initial volume of the array i.e. the values of n and l before the array is extended. And we find that (see section 5.3) extension gain is also an exponential function, hence if n and l increase then the gain increase sharply which is shown in Fig. 11(d). We can conclude that if the initial volume is large then the extension cost for TMA is much higher than that of EKA. Therefore It will be expensive to extend a large array even for small values of (length of extension).

6.3 Overflow

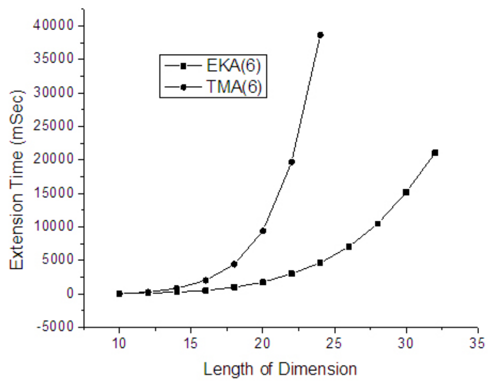
In multidimensional array, the location of an element is calculated using the addressing function described in Section 3. For an n dimensional array with each dimension length = l, maximum value of the coefficient vector can be l^{n-1} which is again multiplied by subscript value (maximum $l-1$). So the resulted value can be written approximately as l^n. This value quickly reaches the machine limit for TMA (eg. for 32 bit machine maximum value can be 232) and thus overflows.



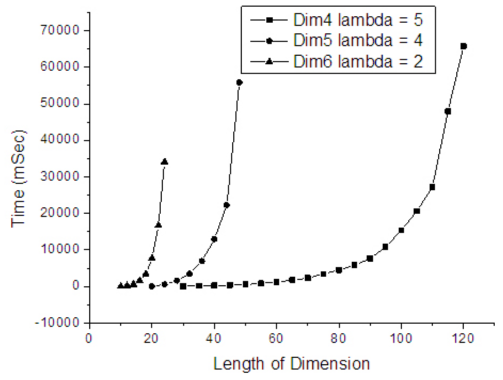
(a) 4D extension Time



(b) 5D extension Time



(c) 6D extension Time



(d) Extension gain

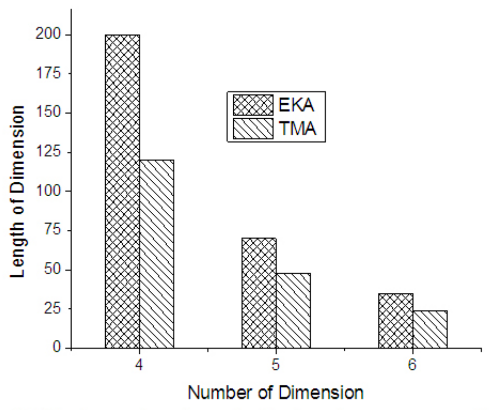
Fig. 11. Extension cost comparison for EKA and TMA.

But in EKA since each of the segments are two dimensional, maximum value will be l^2, which greatly delays the overflow. One more reason is that TMA requires consecutive memory locations up to l^n and hence it overflows soon when l and n is large. On the other hand, in EKA the segments of the subarrays are always two dimensional and distributed. Hence consecutive memory location requirement is less in EKA than TMA. Therefore EKA delays the overflow situation even for large values of l.

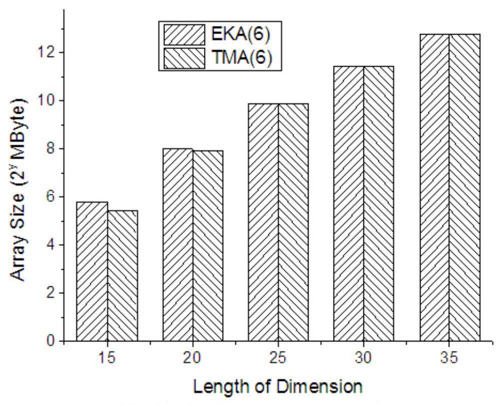
Fig. 12(a) shows the maximum length of dimension reached before overflowing the physical memory. It is found that, $TMA(4)$ reaches a length of 120 each dimension whereas $EKA(4)$ is much greater than that. This is also true for higher dimensional array. Fig.12(b) shows the total storage requirement for EKA and TMA, specifically $n = 6$, for different length of dimension. It is found that both EKA and TMA need almost same amount of storage. In fact, EKA needs slightly higher amount of storage due to its auxiliary tables, but this is very negligible compared to the total requirement. So we can conclude that the nature of storage requirement is almost same for EKA and TMA.

7. CONCLUSION

In this paper, we proposed and evaluated a new scheme namely Extendible Karnaugh Array (EKA) for multidimensional array representation. The main idea of the proposed model is to represent multidimensional array by a set of two dimensional extendible arrays. To extend the TMA dynamically re-linearization is necessary but this is very costly when the array size is large. Therefore we need an array system to extend in all dimensions without costly shuffling of the



(a) Maximum length reached before the occurrence of overflow.



(b) Total storage requirement

Fig. 12. Storage allocation of EKA and TMA.

existing data. Most of array systems do not consider the address space overflow problem. We extend dynamically the multidimensional array and it handles the overflow problem efficiently. We found better results for the proposed model than that of the traditional array representation. This scheme can be successfully applied to database applications especially for multidimensional database or multidimensional data warehousing system. One important future direction of the work is that, the scheme can be easily implemented in parallel platform. Because most of the operations described here is independent to each other. Hence it will be very efficient to apply this scheme in parallel and multiprocessor environment.

REFERENCES

AHSAN, S.M.M., AND HASAN, K.M.A. 2011. An Implementation Scheme for Multidimensional Extendable Array Operations and Its Evaluation. In *proceedings of International Conference on Informatics Engineering & Information Science (pp. 136-150). CCIS 253, Part 5, Springer-Verlag, Berlin, Heidelberg.*

BERTIN, E., AND KIM, W. (1989). Indexing Techniques for Queries on Nested Objects. *IEEE Transactions on Knowledge and Data Engineering, 1(2), 192-214.*

CHEN, Y., DEHNE, F., EAVIS, T., AND CHAPLIN, A.R. (2006). Improved Data Partitioning for Building Large ROLAP Data Cubes in Parallel. *International Journal of Data Warehousing and Mining, 2(1), 1-26.*

CHUN, Y.L., YEH, C. C., AND JEN, S.L. (2003). Efficient Data Compression Method for Multidimensional Sparse Array Operations Based on EKMR Scheme. *IEEE Transactions on Computers, 52(12), 1640-1648.*

CHUN, Y.L., YEH, C.C., AND JEN, S.L. (2003). Efficient Data Parallel Algorithms for Multidimensional Array Operations Based on the EKMR Scheme for Distributed Memory Multicomputer. *IEEE Transactions on Parallel and Distributed Systems, 14(7), 625-639.*

CHUN, Y.L., YEH, C. C., AND JEN, S.L. (2002). Efficient Representation Scheme for Multidimensional Array Operations. *IEEE Transactions on Computers, 51(3), 327-345.*

HASAN, K.M.A. (2009). Compression Schemes for High Dimensional Data for MOLAP. Edited Book, *"Evolving Application Domains of Data Warehousing and Mining: Trends and Solutions", Chapter IV, Information Science Reference, USA.*

HASAN, K.M.A., ISLAM, K., ISLAM, M., AND TSUJI, T. (2009). An Extendible Data Structure for Handling Large Multidimensional Data Sets. In *proceedings of ICCIT, (pp. 669-674). IEEE Explorer.*

HASAN, K.M.A., TSUJI, T., AND HIGUCHI, K. (2007). An Efficient Implementation for MOLAP Basic Data Structure and Its Evaluation. In *proceedings of DASFAA, (pp. 288-299). LNCS 4443. Springer- verlag, Berlin Heidelberg.*

HASAN, K.M.A., TSUJI, T., AND HIGUCHI, K. (2006). A Parallel Implementation Scheme of Relational Tables Based on Multidimensional Extendible Array. *International Journal of Data warehousing and Mining, 2(4), 66-85.*

HASAN, K.M.A., AZUMA, M.N.,TSUJI, T., AND HIGUCHI, K. (2005). An Extendible Array Based Implementation of Relational Tables for Multidimensional Databases. In *proceedings of DaWak. (pp. 233-242). LNCS 3589. Springer- verlag, Berlin Heidelberg.*

KUMAKIRI, M., BEI, L., TSUJI, T., AND HIGUCHI, K. (2006). Flexibly Resizable Multidimensional Arrays. In *proceedings of the 22nd International Conference on Data Engineering Workshops. (pp.83-88). IEEE Computer Society Washington, DC, USA.*

LI, J., AND SRIVASTAVA, J. (2002). Efficient Aggregation Algorithms for Compressed Data Warehouses. *IEEE Transaction on Knowledge and Data Engineering, 14(3), 515-529.*

MANO, M. M. (2005). Digital Logic and Computer Design. *Prentice Hall.*

MARKUS, B., CARSTEN, S., AND GABRIELE, H. (2005). On Schema Evolution in Multidimensional Databases. In *proceedings of DaWak, (pp. 153-164). Springer-Verlag London, UK.*

OTOO, E.J., AND ROTEM, D. (2006). A Storage Scheme for Multi-dimensional Databases Using Extendible Array Files. In *proceedings of the Workshop on STDBM, (pp. 67-76). Seoul, Korea.*

OTOO, E.J., AND ROTEM, D. (2006). Efficient Storage Allocation of Large-Scale Extendible Multi-dimensional Scientific Datasets. In *proceedings of the 18th International Conference on Scientific and Statistical Database Management, (pp. 179-183). IEEE Computer Society Washington, DC, USA.*

OTOO, E. J., AND MERRETT, T.H. (1983). A Storage Scheme for Extendible Arrays. *Computing, 31, 1-9.*

PEDERSEN, T.B., AND JENSEN, C.S. (2001). Multidimensional Database Technology. *IEEE Computer, 34(12), 40-46.*

ROLAND, R.P., AND BAYER, R. (2005). Towards Truly Extensible Database Systems. In *proceedings of DEXA conference. (pp. 596-605). LNCS 3588. Springer-Verlag Berlin Heidelberg*

ROTEM, D., OTOO, E.J., AND SESHADRI, S. (2007). Chunking of Large Multidimensional Arrays. *Lawrence Berkeley National Laboratory, University of California, University of California, LBNL-63230*

ROTEM, D., AND ZHAO, J.L. (1996). Extendible Arrays for Statistical Databases and OLAP Applications. In *proceedings of Scientific and Statistical Database Management, (pp. 108-117). IEEE Computer Society Washington, DC, USA.*

SARAWAGII, S., AND STONEBRAKER, M. (1994). Efficient Organization of Large Multidimensional Arrays. In *Proceedings of ICDE, (pp. 328-336). IEEE Computer Society, Washington, DC, USA.*

SEAMONS, K. E., AND WINSLETT, M. (1994). Physical Schemas for Large Multidimensional Arrays in Scientific Computing Applications. In *Proceedings of SSDBM, (pp. 218-227). IEEE Computer Society, Washington, DC, USA.*

TSUJI, T., KURODA, M., AND HIGUCHI, K. (2008). History Offset Implementation Scheme for Large Scale Multidimensional Data Sets. In *proceedings of ACM Symposium on Applied Computing, (pp.1021-1028). ACM New York, NY, USA.*

ZHAO, Y., DESHPANDE, P.M., AND NAUGHTON, J. F. (1997). An Array Based Algorithm for Simultaneous Multidimensional Aggregates. *Proceedings of the 1997 ACM SIGMOD international conference on Management of data. (pp. 159-170). ACM New York, NY, USA.*

Sk. Md. Masudul Ahsan, born in December, 1980 in Narsingdi, Bangladesh. He received his B.Sc. and M.Sc. degrees in computer science & engineering from Khulna University of Engineering & Technology, Bangladesh in 2003 and 2012 respectively. He is now serving as an assistant professor in the Department of Computer Science and Engineering at Khulna University of Engineering & Technology. His current research interests include database implementation schemes, data warehousing system, visual modeling, and machine vision.

K. M. Azharul Hasan received his B.Sc. (Engg.) from Khulna University, Bangladesh in 1999 and M. E. from Asian Institute of Technology (AIT), Thailand in 2002 both in Computer Science. He received his Ph.D. from the Graduate School of Engineering, University of Fukui, Japan in 2006. His research interest lies in the areas of databases and his main research interests include Data warehousing, MOLAP, Multidimensional databases, Parallel and distributed databases, Parallel algorithms, Information retrieval, Software metric and Software maintenance. He is with the Department of Computer Science and Engineering Khulna University of Engineering and Technology (KUET), Bangladesh since 2001.