

Chunked Extendible Dense Arrays for Scientific Data Storage

G. Nimako, E.J. Otoo, D. Ohene-Kwofie

School of Computer Science
The University of the Witwatersrand
Johannesburg, South Africa

Fifth International Workshop on Parallel Programming Models and Systems Software for High-End Computing (P2S2)

September 2012

- 1 Introduction
- 2 Linear Mapping for a Dense Extendible Array
- 3 Chunking Extendible Dense Arrays
- 4 Axial-Vectors as Memory Resident O_2 -Tree
- 5 Experimental Results
- 6 Summary and Future Work

Introduction

- Multidimensional arrays has been proposed as the most appropriate model for representing scientific databases.
- Scientific data analysis use multidimensional arrays as their fundamental data structure. Examples of **Array Files**:
 - HDF/HDF5 and variants
 - NetCDF/pNetCDF
 - FITS
 - Global Array toolkit
- SciDB is being organised around multidimensional array storage.
- The problem is that such datasets gradually grow to massive sizes of the order of peta-bytes.

Introduction

- Multidimensional arrays has been proposed as the most appropriate model for representing scientific databases.
- Scientific data analysis use multidimensional arrays as their fundamental data structure. Examples of **Array Files**:
 - HDF/HDF5 and variants
 - NetCDF/pNetCDF
 - FITS
 - Global Array toolkit
- SciDB is being organised around multidimensional array storage.
- The problem is that such datasets gradually grow to massive sizes of the order of peta-bytes.

Introduction

- Multidimensional arrays has been proposed as the most appropriate model for representing scientific databases.
- Scientific data analysis use multidimensional arrays as their fundamental data structure. Examples of **Array Files**:
 - HDF/HDF5 and variants
 - NetCDF/pNetCDF
 - FITS
 - Global Array toolkit
- SciDB is being organised around multidimensional array storage.
- The problem is that such datasets gradually grow to massive sizes of the order of peta-bytes.

Introduction

- Multidimensional arrays has been proposed as the most appropriate model for representing scientific databases.
- Scientific data analysis use multidimensional arrays as their fundamental data structure. Examples of **Array Files**:
 - HDF/HDF5 and variants
 - NetCDF/pNetCDF
 - FITS
 - Global Array toolkit
- SciDB is being organised around multidimensional array storage.
- The problem is that such datasets gradually grow to massive sizes of the order of peta-bytes.

Introduction - Problem Motivation

- k -dimensional arrays represented in linear consecutive locations cannot extend without reallocation of already stored elements.

Definition

A realisation of the array $A[\mathbb{U}_0][\mathbb{U}_1]\dots[\mathbb{U}_{k-1}]$ in $L[n]$ for $n = \prod_{j=0}^{k-1} \mathbb{U}_j$, is a mapping function, $\mathcal{F} : \mathbb{U}^k \rightarrow L$, of the elements of A , one-to-one, onto the address, $\{0, 1, \dots, n\}$ with $\mathcal{F}(0, 0, \dots, 0) = 0$.

Row major realisation

$$q = \mathcal{F}(i_0, i_1, i_2, \dots, i_{k-1}) = s_0 + i_0 C_0 + i_1 C_1 + \dots + i_{k-1} C_{k-1}$$

$$C_j = \prod_{r=j+1}^{k-1} \mathbb{U}_r, 0 \leq j \leq k-1, C_{k-1} = 1$$

- The limitation imposed by $\mathcal{F}()$ is that extensions of the array can only be done on one dimension (i.e. that is dimension \mathbb{U}_0 since it was not used in the evaluation of $\mathcal{F}()$).

Introduction - Problem Motivation

- k -dimensional arrays represented in linear consecutive locations cannot extend without reallocation of already stored elements.

Definition

A realisation of the array $A[\mathbb{U}_0][\mathbb{U}_1]\dots[\mathbb{U}_{k-1}]$ in $L[n]$ for $n = \prod_{j=0}^{k-1} \mathbb{U}_j$, is a mapping function, $\mathcal{F} : \mathbb{U}^k \rightarrow L$, of the elements of A , one-to-one, onto the address, $\{0, 1, \dots, n\}$ with $\mathcal{F}(0, 0, \dots, 0) = 0$.

Row major realisation

$$q = \mathcal{F}(i_0, i_1, i_2, \dots, i_{k-1}) = s_0 + i_0 C_0 + i_1 C_1 + \dots + i_{k-1} C_{k-1}$$
$$C_j = \prod_{r=j+1}^{k-1} \mathbb{U}_r, 0 \leq j \leq k-1, C_{k-1} = 1$$

- The limitation imposed by $\mathcal{F}()$ is that extensions of the array can only be done on one dimension (i.e. that is dimension \mathbb{U}_0 since it was not used in the evaluation of $\mathcal{F}()$).

Introduction - Problem Motivation

- This extendibility limitation degrades performance of various array operations particularly in scientific and engineering applications that sometimes undergo interleaved extensions.
- For example, some data processing applications require incremental tiling of adjacent scenes and progressive inclusion of selected bands.
- Extendible arrays, on the other hand can handle dynamic growth in the bounds of the dimensions.
- These arrays can expand in any dimension without reorganising already allocated array element

Outline

- 1 Introduction
- 2 Linear Mapping for a Dense Extendible Array
- 3 Chunking Extendible Dense Arrays
- 4 Axial-Vectors as Memory Resident O_2 -Tree
- 5 Experimental Results
- 6 Summary and Future Work

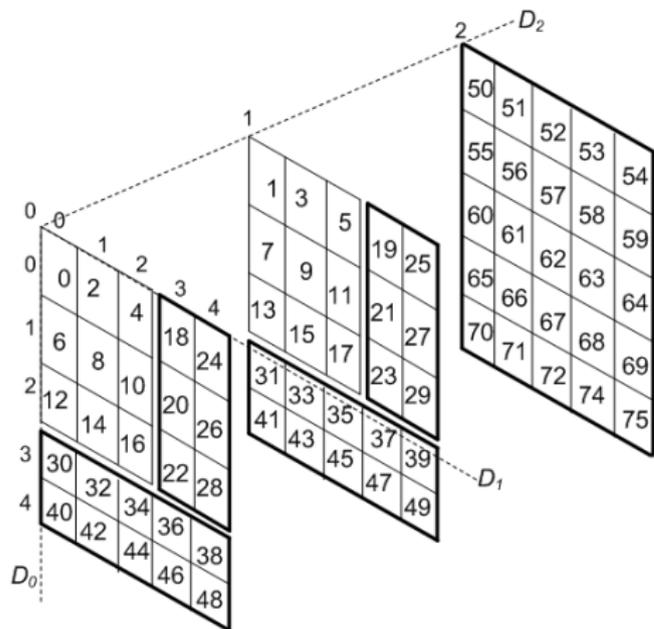
Linear Mapping for a Dense Extendible Array

- The mapping function for extendible array uses axial-vectors to store information needed to compute the function.
- A vector-list of axial-vectors is maintain for each dimension.
- Let $A[\mathbb{U}_0^*][\mathbb{U}_1^*][\mathbb{U}_2^*]$ be an arbitrary 3-dimensional array, where \mathbb{U}_j^* denotes the bound that has the ability to grow as opposed to a fixed bound \mathbb{U}_j as in the conventional array.
- Similarly we employ the notation:
 - $\mathcal{F}()$ when referring to conventional array mapping function.
 - $\mathcal{F}^*()$ when referring to a mapping function that allows extendibility in any dimension

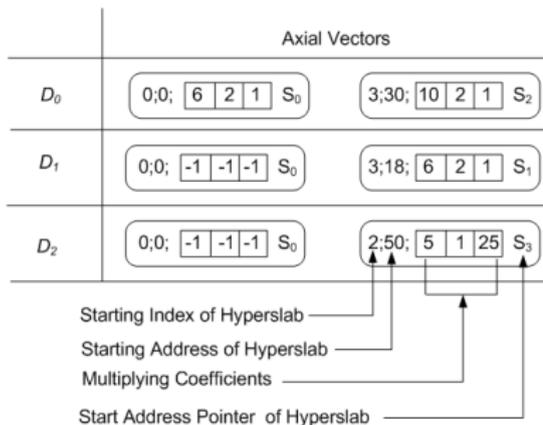
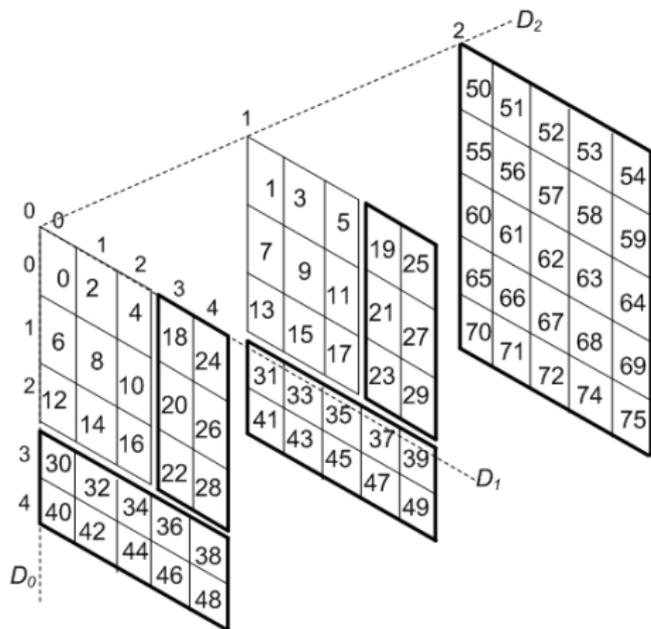
Linear Mapping for a Dense Extendible Array

- The mapping function for extendible array uses axial-vectors to store information needed to compute the function.
- A vector-list of axial-vectors is maintain for each dimension.
- Let $A[\mathbb{U}_0^*][\mathbb{U}_1^*][\mathbb{U}_2^*]$ be an arbitrary 3-dimensional array, where \mathbb{U}_j^* denotes the bound that has the ability to grow as opposed to a fixed bound \mathbb{U}_j as in the conventional array.
- Similarly we employ the notation:
 - $\mathcal{F}()$ when referring to conventional array mapping function.
 - $\mathcal{F}^*()$ when referring to a mapping function that allows extendibility in any dimension

Linear Mapping for a Dense Extendible Array - Illustration



Linear Mapping for a Dense Extendible Array - Illustration



Linear Mapping for a Dense Extendible Array

- Suppose that in a k -dimensional extendible array $A[\mathbf{U}_0^*][\mathbf{U}_1^*][\mathbf{U}_2^*]\dots[\mathbf{U}_{k-1}^*]$, dimension l is extended by λ_l , then the index range increases from \mathbf{U}_l^* to $\mathbf{U}_l^* + \lambda_l$.
- Let the location $A\langle 0, 0, \dots, \mathbf{U}_l^*, \dots, 0 \rangle$ (i.e. the starting location of an allocated *hyperslab*) be denoted as $\ell_{\mathbf{Z}_l^*}$ where $\mathbf{Z}_l^* = \prod_{r=0}^{k-1} \mathbf{U}_r^*$.

The Mapping Function

$$q^* = \mathcal{F}^*(\langle i_0, i_1, i_2, \dots, i_{k-1} \rangle) = \mathbf{Z}_{\mathbf{U}_l^*}^0 + (i_l - \mathbf{U}_l^*)C_l^* + \sum_{\substack{j=0 \\ j \neq l}}^{k-1} i_j C_j^*$$

$$C_l^* = \prod_{\substack{j=0 \\ j \neq l}}^{k-1} \mathbf{U}_j^*$$

$$C_j^* = \prod_{\substack{r=j+1 \\ r \neq l}}^{k-1} \mathbf{U}_r^*$$

Linear Mapping for a Dense Extendible Array

- Suppose that in a k -dimensional extendible array $A[\mathbf{U}_0^*][\mathbf{U}_1^*][\mathbf{U}_2^*]\dots[\mathbf{U}_{k-1}^*]$, dimension l is extended by λ_l , then the index range increases from \mathbf{U}_l^* to $\mathbf{U}_l^* + \lambda_l$.
- Let the location $A\langle 0, 0, \dots, \mathbf{U}_l^*, \dots, 0 \rangle$ (i.e. the starting location of an allocated *hyperslab*) be denoted as $\ell_{\mathbf{Z}_l^*}$ where $\mathbf{Z}_l^* = \prod_{r=0}^{k-1} \mathbf{U}_r^*$.

The Mapping Function

$$q^* = \mathcal{F}^*(\langle i_0, i_1, i_2, \dots, i_{k-1} \rangle) = \mathbf{Z}_{\mathbf{U}_l^*}^0 + (i_l - \mathbf{U}_l^*)C_l^* + \sum_{\substack{j=0 \\ j \neq l}}^{k-1} i_j C_j^*$$

$$C_l^* = \prod_{\substack{j=0 \\ j \neq l}}^{k-1} \mathbf{U}_j^*$$

$$C_j^* = \prod_{\substack{r=j+1 \\ r \neq l}}^{k-1} \mathbf{U}_r^*$$

Outline

- 1 Introduction
- 2 Linear Mapping for a Dense Extendible Array
- 3 Chunking Extendible Dense Arrays**
- 4 Axial-Vectors as Memory Resident O_2 -Tree
- 5 Experimental Results
- 6 Summary and Future Work

Chunking Extendible Dense Arrays

- The use of the vector-list for axial-vectors can be expensive and depends particularly on the interruptible expansions (*cubical extensions*).
- Such interruptible expansion causes the addition of a new entry in the vector-list.
- Chunking the array gives some additional advantages:
 - It gives contiguous storage allocations for the elements of the chunks.
 - When arrays are allocated onto secondary storage, I/O can be made in multiples of the chunk size.
- The allocation is done in *chunks* as opposed to the single elements.

Chunking Extendible Dense Arrays

- The use of the vector-list for axial-vectors can be expensive and depends particularly on the interruptible expansions (*cubical extensions*).
- Such interruptible expansion causes the addition of a new entry in the vector-list.
- Chunking the array gives some additional advantages:
 - It gives contiguous storage allocations for the elements of the chunks.
 - When arrays are allocated onto secondary storage, I/O can be made in multiples of the chunk size.
- The allocation is done in *chunks* as opposed to the single elements.

Chunking Extendible Dense Arrays

- Given a chunked block $Q[\chi_0][\chi_1][\chi_2]\dots[\chi_{k-1}]$, the number of chunk indices, ρ_i for a given dimension i , is given by:

$$\rho_i = \left\lceil \frac{\mathbf{U}_i^*}{\chi_i} \right\rceil$$

- The allocation of chunks, denoted by A_c , becomes $A_c[\rho_0][\rho_1][\rho_2]\dots[\rho_{k-1}]$.

- An entry is made to the requisite axial-vector only if this condition is met:

$$[\mathbf{U}_i^* + \lambda_i] > [\rho_i \times \chi_i]$$

- The number of chunks ρ_i to be allocated is given by:

$$\rho_i = \left\lceil \frac{[\mathbf{U}_i^* + \lambda_i] - [\rho_i \times \chi_i]}{\chi_i} \right\rceil$$

Chunking Extendible Dense Arrays

- Given a chunked block $Q[\chi_0][\chi_1][\chi_2]\dots[\chi_{k-1}]$, the number of chunk indices, ρ_i for a given dimension i , is given by:

$$\rho_i = \left\lceil \frac{\mathbf{U}_i^*}{\chi_i} \right\rceil$$

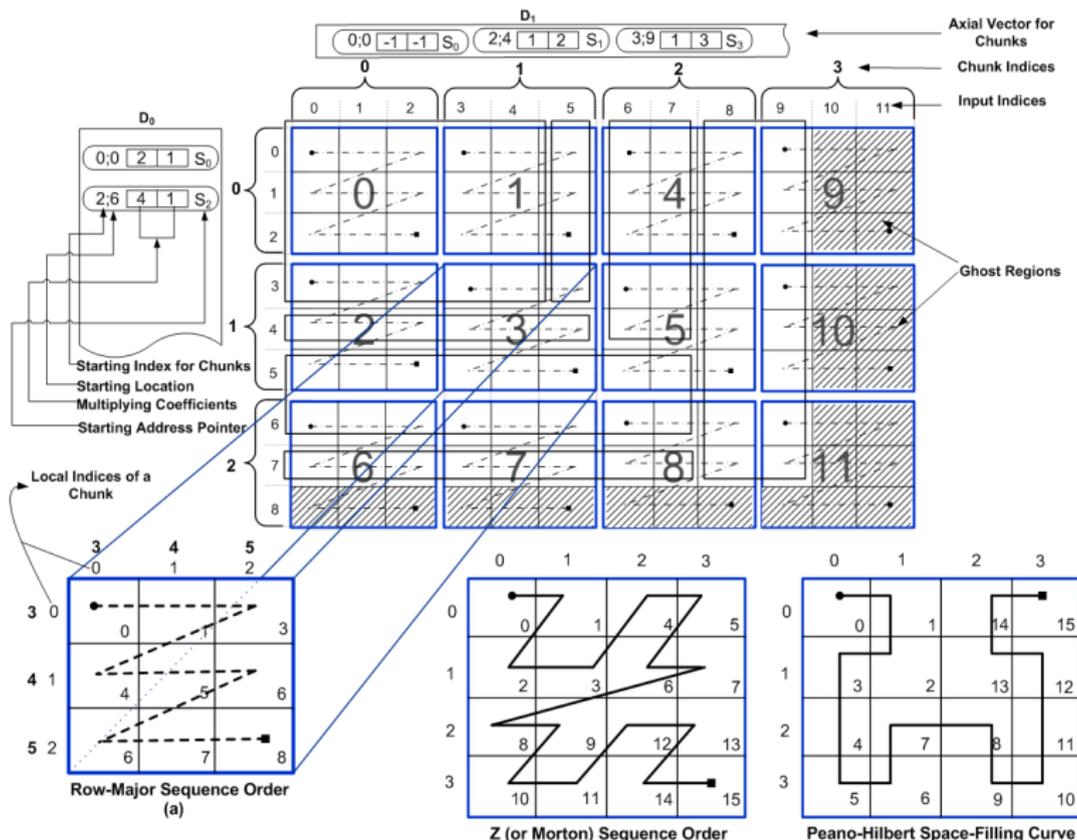
- The allocation of chunks, denoted by A_c , becomes $A_c[\rho_0][\rho_1][\rho_2]\dots[\rho_{k-1}]$.
- An entry is made to the requisite axial-vector only if this condition is met:

$$[\mathbf{U}_i^* + \lambda_i] > [\rho_i \times \chi_i]$$

- The number of chunks ρ_i to be allocated is given by:

$$\rho_i = \left\lceil \frac{[\mathbf{U}_i^* + \lambda_i] - [\rho_i \times \chi_i]}{\chi_i} \right\rceil$$

Chunking Extendible Dense Arrays



Chunking Extendible Dense Arrays

- To access an array element $A\langle i_0, i_1, i_2, \dots, i_{k-1} \rangle$, the *input indices* $\langle i_0, i_1, i_2, \dots, i_{k-1} \rangle$ is translated into *chunk indices* $\langle j_0, j_1, j_2, \dots, j_{k-1} \rangle$ where

$$j_i = \left\lfloor \frac{i_i}{\chi_i} \right\rfloor$$

- The starting address, q_c^* of the chunk containing $A\langle i_0, i_1, i_2, \dots, i_{k-1} \rangle$ can be found by:

The Mapping Function for Chunked Extendible Array

$$q_c^* = \mathcal{F}^*(\langle j_0, j_1, j_2, \dots, j_{k-1} \rangle) = Z_{\rho_l}^0 + (j_l - \rho_l) C_l^* + \sum_{\substack{m=0 \\ m \neq l}}^{k-1} j_m C_m^*$$

$$C_l^* = \prod_{\substack{m=0 \\ m \neq l}}^{k-1} \rho_m$$

$$C_m^* = \prod_{\substack{r=m+1 \\ r \neq l}}^{k-1} \rho_r$$

Chunking Extendible Dense Arrays

- To access an array element $A\langle i_0, i_1, i_2, \dots, i_{k-1} \rangle$, the *input indices* $\langle i_0, i_1, i_2, \dots, i_{k-1} \rangle$ is translated into *chunk indices* $\langle j_0, j_1, j_2, \dots, j_{k-1} \rangle$ where

$$j_i = \left\lfloor \frac{i_i}{\chi_i} \right\rfloor$$

- The starting address, q_c^* of the chunk containing $A\langle i_0, i_1, i_2, \dots, i_{k-1} \rangle$ can be found by:

The Mapping Function for Chunked Extendible Array

$$q_c^* = \mathcal{F}^*(\langle j_0, j_1, j_2, \dots, j_{k-1} \rangle) = Z_{\rho_l}^0 + (j_l - \rho_l) C_l^* + \sum_{\substack{m=0 \\ m \neq l}}^{k-1} j_m C_m^*$$

$$C_l^* = \prod_{\substack{m=0 \\ m \neq l}}^{k-1} \rho_m$$

$$C_m^* = \prod_{\substack{r=m+1 \\ r \neq l}}^{k-1} \rho_r$$

Chunking Extendible Dense Arrays

- To compute the address of $A\langle i_0, i_1, i_2, \dots, i_{k-1} \rangle$ within the *local chunk*, the input indices $\langle i_0, i_1, i_2, \dots, i_{k-1} \rangle$ needs to be translated to local chunk indices $\langle i_{c0}, i_{c1}, i_{c2}, \dots, i_{c(k-1)} \rangle$ by :

$$i_{cm} = (i_m \bmod \chi_m)$$

- The address of $A\langle i_0, i_1, i_2, \dots, i_{k-1} \rangle$ is only a displacement within the chunk.
- This can be done by using a row-major sequence order or column-major order.
- If the chunk size is 2^n where $n \geq 2$, then the **Z-order sequence** or **Peano-Hilbert space filling curve** can be used.

Chunking Extendible Dense Arrays

- To compute the address of $A\langle i_0, i_1, i_2, \dots, i_{k-1} \rangle$ within the *local chunk*, the input indices $\langle i_0, i_1, i_2, \dots, i_{k-1} \rangle$ needs to be translated to local chunk indices $\langle i_{c0}, i_{c1}, i_{c2}, \dots, i_{c(k-1)} \rangle$ by :

$$i_{cm} = (i_m \quad \text{mod} \quad \chi_m)$$

- The address of $A\langle i_0, i_1, i_2, \dots, i_{k-1} \rangle$ is only a displacement within the chunk.
- This can be done by using a row-major sequence order or column-major order.
- If the chunk size is 2^n where $n \geq 2$, then the **Z-order sequence** or **Peano-Hilbert space filling curve** can be used.

Outline

- 1 Introduction
- 2 Linear Mapping for a Dense Extendible Array
- 3 Chunking Extendible Dense Arrays
- 4 Axial-Vectors as Memory Resident O_2 -Tree**
- 5 Experimental Results
- 6 Summary and Future Work

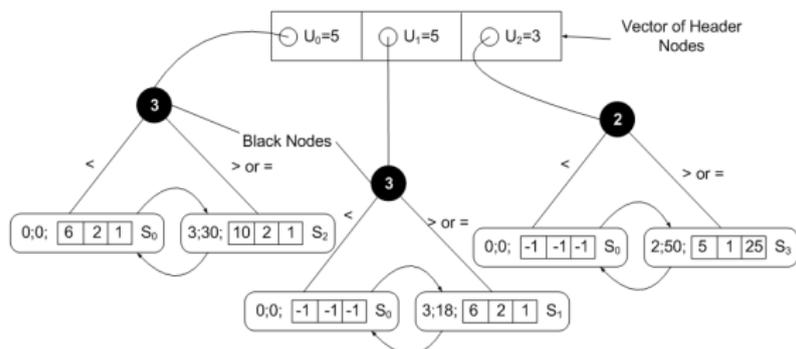
Axial-Vectors as Memory Resident O_2 -Tree

- A new approach to maintaining these axial-vectors in memory is with the use of O_2 -Tree.
- An O_2 -Tree is an augmented Red-Black Tree with data records stored only at the leaf nodes.
- A metadata file F_m stores the records that correspond to the leaf nodes of the O_2 -Tree.
- These records in F_m are used to reconstruct the memory resident O_2 -Tree.

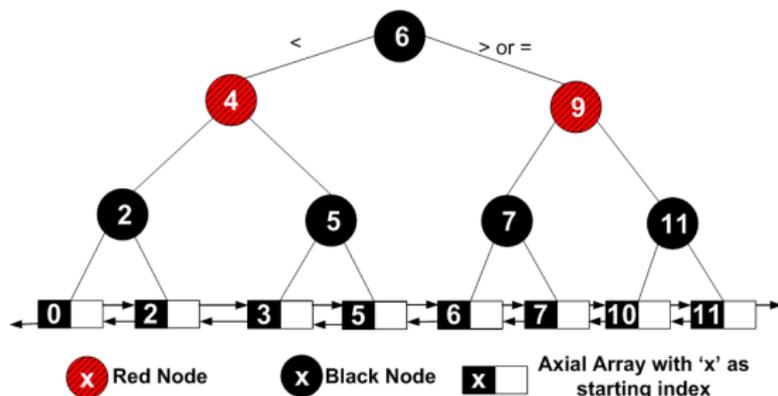
Axial-Vectors as Memory Resident O_2 -Tree

- A new approach to maintaining these axial-vectors in memory is with the use of O_2 -Tree.
- An O_2 -Tree is an augmented Red-Black Tree with data records stored only at the leaf nodes.
- A metadata file F_m stores the records that correspond to the leaf nodes of the O_2 -Tree.
- These records in F_m are used to reconstruct the memory resident O_2 -Tree.

Axial-Vectors as Memory Resident O_2 -Tree



General Structure of the O_2 -Tree :

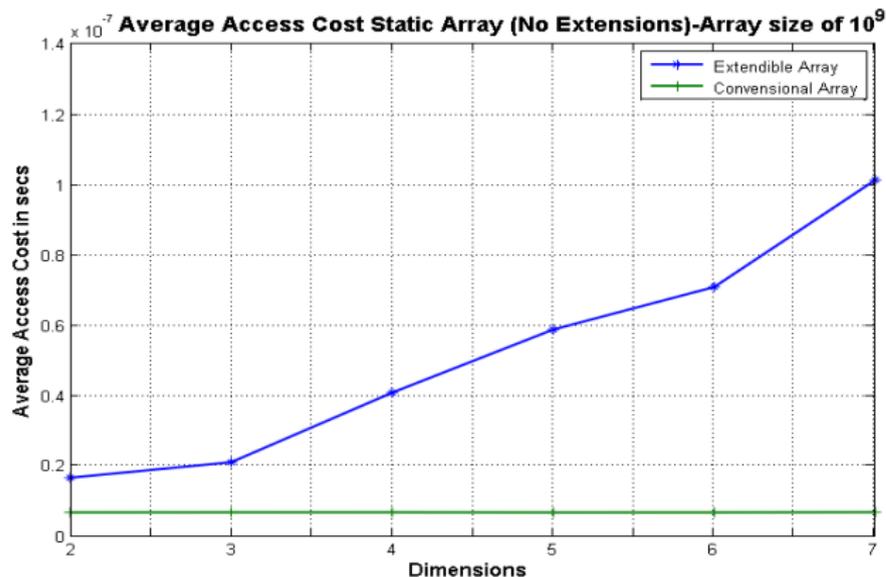


Outline

- 1 Introduction
- 2 Linear Mapping for a Dense Extendible Array
- 3 Chunking Extendible Dense Arrays
- 4 Axial-Vectors as Memory Resident O_2 -Tree
- 5 Experimental Results**
- 6 Summary and Future Work

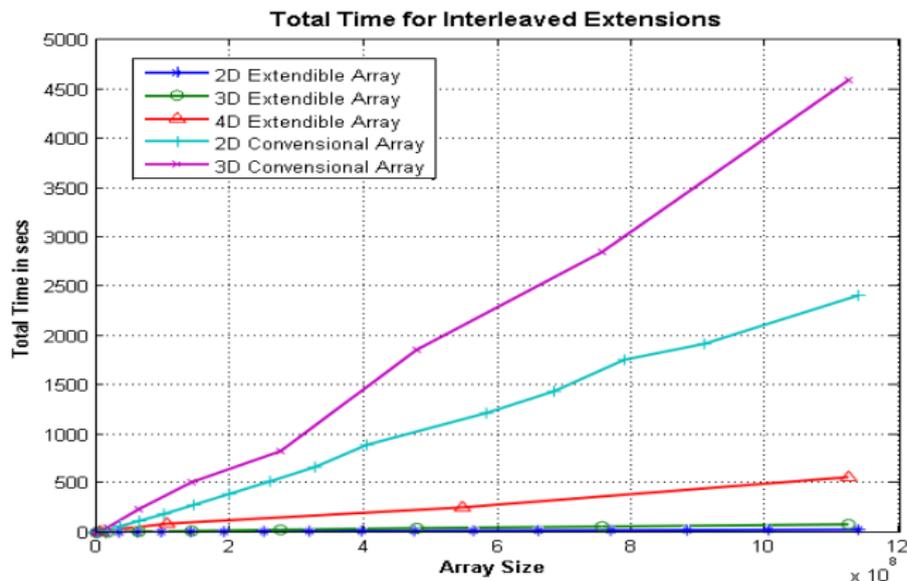
Experimental Results

- Average Access Cost without Extensions (in Memory)



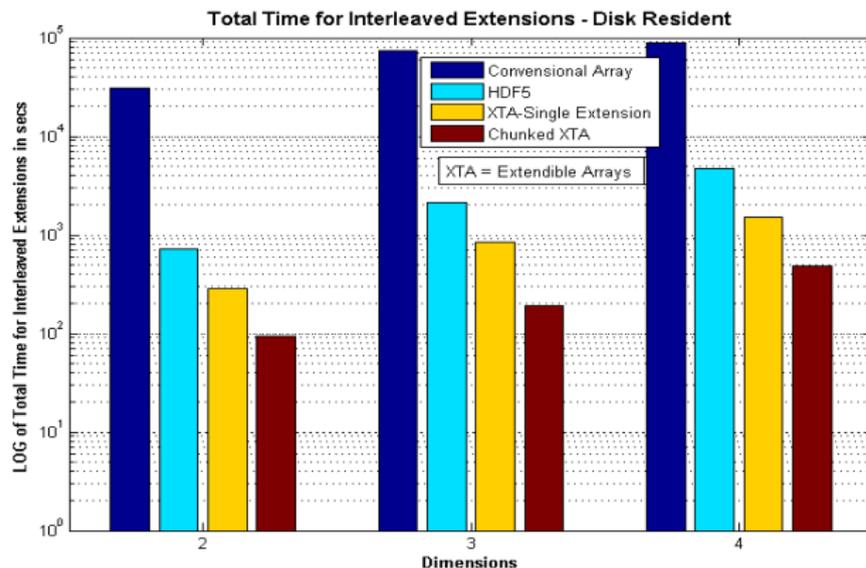
Experimental Results

- Total Access Cost for Interleaved Extensions in Memory



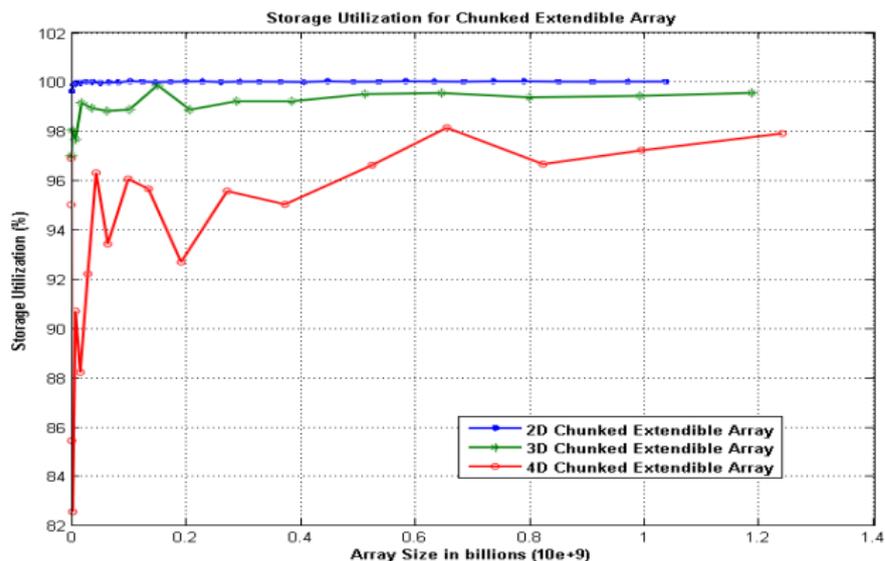
Experimental Results

- Total Access Cost for Interleaved Extensions on Disk



Experimental Results

- Storage Utilization for Chunked Extendible Array



Outline

- 1 Introduction
- 2 Linear Mapping for a Dense Extendible Array
- 3 Chunking Extendible Dense Arrays
- 4 Axial-Vectors as Memory Resident O_2 -Tree
- 5 Experimental Results
- 6 Summary and Future Work

Summary and Future Work

- In this paper, we have given an implementation of the chunked extendible dense arrays.
- By chunking the elements of the array, the chunked extendible array can be conveniently stored in files.
- Array elements are then accessed into and out of memory in multiples of chunks with the aid of a mapping function.
- The organisation of extendible arrays using such a mapping function is highly appropriate for most scientific datasets where the model of the data is perceived to be in the form of large array files.
- Currently the appropriate APIs for integrating our scheme with the Global Array Toolkit are being developed.