Dope vectors The real organizer of memory management for arrays

Ankita A. Dongre
Department of Masters of Computer Applications
G. S. Mandal's Marathwada Institute of Technology, Aurangabad, India

Abstract- An array is a named container object that holds value of single type. The array can be of type static as well as dynamic depending upon the usage in any particular application. For static arrays the array handling mechanism is quite simple in comparison with dynamic arrays. This paper aims to give an insight towards the most effective data structure implementation called an Array. Dope Vectors typically work behind the scenes for accessing, storing and implementation of arrays in memory regardless of their type playing important role in memory management with the collaboration of language implementation specifications. They hold the information about dynamic arrays as their key attributes such as lower bound of array, size of every element along with upper bond of a data structure object. This makes the implementation of dope vectors as a major memory handling resource to the operating system with the dynamic structures as arrays and records aiming towards a feasible and efficient structure that lead to development of most powerful data structure called as Link Lists.

Keywords – Dope Vector (DV), Descriptor, Virtual Origin (VO), l value.

I. INTRODUCTION

In Early programming languages like f9 and early versions of C, dope vector was designed as a struct DopeVector and which is being used by the system program called as debugger to decode the information present in that vector via a link. This was the case with most of the dynamically allocated arrays (whose size is not known at compile time). Whereas in case of statically allocated arrays virtual origin, lower and upper bounds of every subscript and total size of array component was known at compile time only and this information was stored in dope vector structure.

VO(Virtual Origin)
Lower bound for subscript 1
Upper bound for subscript 1
. . . .
Lower bound for subscript n
Upper bound for subscript n
Size of Component

FIG: 1.1 DESCRIPTOR OF DOPE VECTOR

However accessing the array elements with descriptors also have certain points to take care of:

1) Descriptors should be defined correctly to access the exact and correct memory locations to access the elements of array.

2) Codes that uses array descriptors should not be portable[1] to other compilers and platforms.

3) Descriptors if already developed by compiler can not update or modify their format.

The array pointers and an allocatable arrays uses descriptor whereas explicit shape[2] and assumed size arrays[3] do not use descriptor.

A **Dope Vector** (DV) can be defined as a data structure, used to hold information about a data object[4]. E.g. an array [5].

A dope vector consists following information about array.

1) Type of array: it could be primitive or reference.
2) Rank of array: number of subscripts or dimensions of that array.
3) Extends of array [6]: number of elements in that dimension.
4) Stride of array: number of bytes one has to travel to navigate from one array element to another element.
5) Pointer to the block in memory which contains array elements. (It can store the base address of array or address of element which is subscripted to zero.)

II. ACCESSING ARRAYS USING DOPE VECTORS

Let us examine with a simple example as below:

Vector **A** and the size of each component say **E** are kwon at compile time. If we wish to access the **I**th element of the vector A then the **lvalue** [7] of Ith element considering it at a initial element can be calculated as (I-1) × E memory locations. In addition lower bound of vector A can be denoted as **LB** and the number of elements to be skipped can be calculated with (I – LB) instead of (I-1), if I is at some other location in memory. At such time the initial location can be represented by α and the accessing formulae for vector A takes a form of:

lvalue (A[I])= $\alpha + (I - LB) \times E$

which again can be calculated as

lvalue (A[I])= $(\alpha - LB \times E) + (I \times E)$.[8] ....[1]

lvalue (A[I]) = $\alpha - LB \times E$

In above scenario for vector A storage is allocated once then the value of $(\alpha - LB \times E)$ in equation 1 can be replaced by a constant K and the formulae takes a new form as:

lvalue (A[I])= $K + (I \times E)$

Languages like fortran, Pascal follow the same schema. Considering language C with a char array having memory size 1 byte for each element and the LB which is always 0 the above equation 1 reduces to:

lvalue (A[I]) = $\alpha + I$

Let us now consider accessing the array element at the subscript location 0 i.e. value of I is 0. Replacing it in equation 1 the lvalue can be calculated as:

lvalue (A[I])= $(\alpha - LB \times E)$

and $(\alpha - LB \times E)$ is equivalent to K so the new expression is

lvalue= K.

This is the virtual origin of vector A which yields array accessing methodology and algorithm generation of vector components. The virtual origin also keeps track of generation of vector storage with the mathematical formulae $D + N \times E$ memory locations. Where D is size of descriptor, N is size of vector and E is size of each component. The VO can be finally calculated by the formulae $VO = \alpha - LB \times E$. And the elements can be accessed with the help of lvalue (A[I])= $VO + I \times E$.

III. STORING ARRAY ELEMENTS USING DOPE VECTORS

Two different arrays sharing the same storage could be defined by having the first pointer pointing to the same vector. Two arrays having the same shape and size could use the same dope vector by having the second pointers same. The typical storage of elements can be structured as follows:

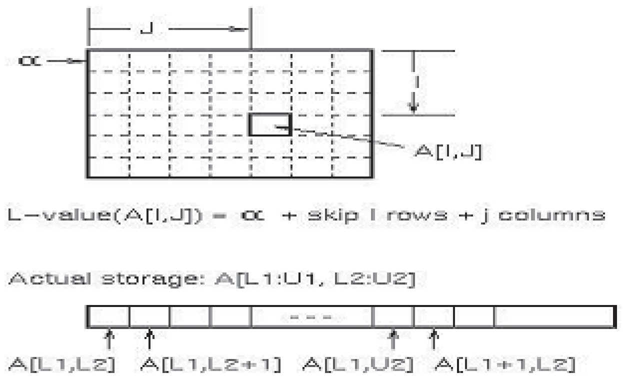
J
α
I
A[I,J]
L−value(A[I,J]) = α + skip I rows + j columns
Actual storage: A[L1:U1, L2:U2]
- - -
A[L1,L2]
A[L1,L2+1]
A[L1,U2]
A[L1+1,L2]

Fig. 1.2 STORING ARRAY ELEMENTS IN MEMEORY

IV. IMPLEMENTATION OF ARRAYS USING DOPE VECTORS

During compilation information about arrays are stored in dope vectors and the VO and descriptors of arrays together manage the handling mechanism of arrays. The dope vector can have the pointers with the descriptors to access the subscript of array. The number of pointers totally depends upon the number of dimensions of the array. By default Fortran starts indexing with 1 whereas languages like C, C++ even JAVA starts indexing with 0. In such cases the implementation of arrays differs with parsing techniques. Talking specifically, with multidimensional arrays C moves with its right index and Fortran moves with its left index frequently. For example say array A is two dimensional array and the declaration of first four elements in Fortran provided by IEEE standardization is A (1 , 1) , A (2 , 1) , A (3 , 1) , A (1 , 2) and by C language is A [0] [0] , A [0] [1] , A [0] [2] , A [1] [0]. This example clears the implementation of arrays with dope vectors as a subscript navigation technique.

As we know by now that dope vector is used to store the information about the data structures especially dynamic arrays along with its memory layouts and general information like rank of arrays, strides of array, extends of arrays, type of array etc. It also includes identifier (user name) address of parent as well as address of child element. As they are memory address they are generally register offsets or core references to internal memory. So the dope vectors loads and stores the run time information of objects effectively.

V. CONCLUSION

As studied earlier, one of the key components of dope vector is the link it holds for memory location serving extreme ease for memory to pass entire array to procedures in high level programming languages like fortan, C, C++ etc. As well as before link lists the internal structure of computer memory was organized by dope vectors to keep track of various memory locations used and engaged with different applications. Here by we can conclude that perception towards arrays is not only limited to accessing and storing multiple elements under a single head but also extends to accessing and storing multiple memory locations through a single structure called as Dope Vector. In today's era the Dope Vectors hold a special status bit to know the system about the activeness of itself, if the Dope Vector is not active it would be reallocated when needed. In this way systems could achieve the memory management more efficiently.

REFERENCES

[1] http://www.oxforddictionaries.com/definition/english/portable

[2]]http://publib.boulder.ibm.com/infocenter/comphelp/v7v91/index.jsp?topic=%2Fcom.ibm.xlf91a.doc%2Fxlflr%2Fexplshp.htm

[3] http://publib.boulder.ibm.com/infocenter/comphelp/v7v91/index.jsp?topic=%2Fcom.ibm.xlf91a.doc%2Fxlflr%2Fexplshp.htm

[4] Pratt T. and M. Zelkowitz, Programming Languages: Design and Implementation (Third Edition), Prentice Hall, Upper Saddle River, NJ, (1996) pp 114

[5] http://msdn.microsoft.com/en-us/library/ms379570(v=vs.80).aspx

[6] Extends of array is calculated as the value of upper bound minus value of lower bound plus one.

[7] http://msdn.microsoft.com/en-us/library/f90831hc.aspx

[8] http://books.google.co.in/books?id=Q76TM8CKr2EC&pg=PA200&lpg=PA200&dq=what+is+virtual+origin+in+dope+vector&source=bl&ots=h9wrNl0T5p&sig=ZpLLVwYsGX2DcbxFH1TAV_8cZjM&hl=en&sa=X&ei=RtebU5uwBYW_uATtpYLADA&ved=0CD8Q6AEwBg#v=onepage&q=what%20is%20virtual%20origin%20in%20dope%20vector&f=false