The Distributed Array Descriptor for a PCRC HPF Compiler

Version 2.0

SCCS-770d

Bryan Carpenter, James Cowie*, Don Leskiw, and Xiaoming Li†

Northeast Parallel Architectures Center,
Syracuse University,
111 College Place,
Syracuse, New York, 13244-4100

Octobor 11, 1996

Abstract

We describe a *distributed array descriptor* that can be used by a runtime supporting HPF-like compilers. This descriptor captures all five types of alignment and BLOCK and CYCLIC distribution as defined in HPF specification. In essence, this descriptor does not distinguish whole array and array sections.

Prior to this version, we had versions 1.0, 1.1, and 1.2. This version is not only an update of previous versions, but more importantly it also directly reflects our current practice in an HPF compilation effort.

Introduction

This document is the first of two in a sequence; it will be followed by the *Common Runtime System Interface (CRSI)*. The CRSI will specify a common runtime library for HPF, and will include various collective communication primitives, standard mapping inquiry functions, and transformational intrinsic functions.

In preparation for the CRSI, this document specifies a simple data descriptor for distributed arrays to be used by CRSI runtime functions called from SPMD node programs. We call this specification the **Distributed Array Descriptor** or **DAD**.

The DAD's primary role is to provide a standard format for arguments to common run-time system functions. By conforming to the DAD specification when describing arrays and

*Cooperating Systems Corporation, *http://www.cooperate.com*
†Visiting from Harbin Institute of Technology of China

array sections, compiler writers can take advantage of the common functionality of any CRSI implementation.

We suspect that the DAD specified in this document might eventually be used more generally, as a general-purpose data descriptor for arrays and array sections passed between user subroutines, and to describe distributed arrays created in language contexts other than HPF.

Overview

The DAD is a simple, flat table of data which describes a mapping from HPF's global array space to the local array space in each SPMD node program. When an array section is used as an argument to a subprogram of the node program, a new DAD is created and passed.

The DAD structure will be visible from both C (C++) and Fortran SPMD programs. DAD supports both C-style and Fortran-style local arrays, by including selectable row/column majority.

In the following sections, we first briefly define some terms; then we state the DAD requirements, and describe the specific members (or "slots") of a Distributed Array Descriptor before concluding with some examples.

Terminology

Many of the terms we use to describe the Distributed Array Descriptor will be familiar to HPF programmers and compiler implementors. An **index** designates a specific array element, while an **address** designates a physical memory location. The arrays described in an HPF program are **global arrays**; the arrays described in a SPMD node program are **local arrays. Global indices** designate elements of global arrays, while **local indices** designate elements of local arrays. Each array in an HPF program will have, explicitly or implicitly, a **template** that it is aligned with. There are five types of alignment between an array and its template, namely, **offset** alignment, **strided** alignment, dimension **permuted** alignment, **collapsed** alignment, and **replicated** alignment. As a result of alignment, not all elements of the local array correspond to elements in the global array from which they are derived (we assume template shape, instead of HPF array shape, determines allocation of local arrays); those which do we call **effective elements** of the local array. There are three types of distribution modes by which a template may be distributed onto a processor grid, namely **block** distribution, **cyclic** distribution, and **collapsed** distribution. Although the DAD does not support explicit parameterized distributions (i.e., BLOCK(n) or CYCLIC(n)), it does support so-called **rank reduced sectioning**, which can be viewed as some kind of BLOCK(n) or CYCLIC(n).

A **regular array section** selects certain elements of an array by supplying a lower bound (or **section offset**), an upper bound, and a **section stride** per dimension. We will therefore use the generic term "array" to encompass both arrays and array sections. An **array slice** is an array expression containing scalar subscripts. In Fortran, an array slice is considered to have a reduced rank equal to the number of nonscalar subscripts.

Requirements

Global arrays are distributed by HPF directives. The complexity of the mapping between the global and local arrays determines the complexity of the distributed array descriptor (DAD). The DAD should precisely record how an array is distributed by allowable language elements/directives, in such a way that the HPF compiler can conveniently generate it and the runtime can efficiently use it to carry out collective computation and communication over the distributed arrays.

Thus, in our case, a DAD will capture all five possible types of alignment between an array and its template [1], and describe how that template is distributed. The DAD must support (1) effective mapping between global array indices to local array indices, (2) efficient iteration over the local elements of an array, and (3) a mechanism for passing array information to a variety of data movement functions.

[1] ... with one important assumption: if array alignment is replicated along certain dimensions of a template, the corresponding dimensions of the template must be distributed, and the extents of the dimensions must be equal to the extents of the corresponding dimensions of the processor grid. Otherwise, the rank of the local array may not equal that of the global array.

Figure 1: A suggested layout of DAD contents

The Distributed Array Descriptor

The DAD is a collection of parameters which are sufficient to describe distributed arrays and regular array sections to be passed to the common runtime system functions of the CRSI. We group these parameters into *per-dimension* attributes and *per-array* attributes. In our DAD, the `DIM` structure (summarized in Table **??**) describes a single-dimension mapping between an array and its distributed template. In addition, the DAD contains a few attributes which are specified once per array, rather than once per dimension; these are shown in Table **??**.

Group	*Field*	*Interpretation*
Global	g_extent	extent of global array/section in this dimension
	t_extent	extent of template this array dim is aligned with.
	t_stride	stride on template, undefined for t_extent=0
	t_offset	offset on template, undefined for t_extent=0
	dist_code	template distribution code
	on_pdim	processors dimension this array dim is distributed on
Local	l_extent	local array shape (as allocated) in this dimension
	l_lb	local index of first effective element in this dim.
	l_ub	local index of last effective element in this dim
	l_stride	local stride in memory in this dimension
	ghost_size	size of ghost area at both ends of this dimension
Group	p_shape	# processors in this dim of the processor grid
	local_coord	coordinate of this processor in the grid
	slice_coord	coordinate of the sub grid holding this array.

Table 1: DIM: structure of per-dimension information

Type	*Field*	*Interpretation*
void*	base_address	base address of local array
int	element_type	code for the element data type.
int	rank	number of dimensions of the array
int	p_rank	rank of the processor grid
int	comm	a handle to the processor grid
int	majority	1 for Fortran and `rank` for C.

Table 2: DAD: structure of per-array information

Observing Fortran allows up to 7 dimensions for arrays, the information above may be conveniently collected in a 15x7 matrix plus a separate entity for base address, as show in Figure **??**.

While some of the slots are obvious, others are subtle, and some are redundant, because the DAD requirements for high performance and flexibility conflicted with the desire to construct a perfectly minimal, precisely orthogonal descriptor standard. For clarity, we give further elaboration of each slot below, followed by some examples.

Most generally, the DAD describes a regular section of an array, which is aligned to a template, which is distributed over processors. All other cases (unsectioned (whole) arrays, arrays with default templates, etc.) are degenerate cases of this most general case.

- The *global portion* of the DAD describes the array section and its mapping parameters. These are quantities collected from application program, independent of processors. This portion is mainly useful for global/local index mapping.
- The *local portion* of the DAD is derived from the global portion for efficiency. They directly describe physical data layout in local memory. Some of the items in this portion are processor dependent. This portion is mainly useful for traversal of array elements.
- The *group portion* of the DAD describes the processor group that owns this array section. Rank reduced array sectioning is supported. This portion is useful for knowing who should participate in some collective operations.

Per-dimension information

g_extent: Extent of global array/section in this dimension. Note only extent is recorded, which implies compiler should normalize parameters when calling runtime function for global/local index mapping.

t_extent: Extent of template this array dimension is aligned with, `t_extent=0` if this array dimension is collapsed in ALIGN directive. For convenience and w.l.o.g, we always adjust the extent to the closest multiple of number of processors this template dimension is distributed on. The possible permutation between array dimensions and template dimensions during alignment is taken care of by this arrangement, i.e., compiler would provide appropriate `t_extent` for DAD constructor.

t_stride: Stride on template, undefined (denoted by -1) for t_extent=0. Initially, it is the alignment stride as given in application program. At runtime, it may be modified due to sectioning.

t_offset: Offset on template, undefined (denoted by -1) for t_extent=0. This is *not* alignment stride as in ALIGN directive. Instead, it's always the relative position of the first array element on the template, starting from 0.

dist_code: Distribution code, < 0 for block, 0 for collapsed, > 0 for cyclic. While the coding scheme seems arbitrary, we do see one advantage of this one later.

on_pdim: Processors dimension this array dimension is distributed on.

l_extent: Local extent of this dimension in node program. This reflects actual memory allocation, including possible ghost area.

l_lb: Index of the first effective element in local array in this dimension.

l_ub: Index of the last effective element in local array in this dimension. If l_ub is less than l_lb, no effective element in this processor. Usually, `l_lb` and `l_ub` are processor dependent.

l_stride: Local stride in this dimension. `l_lb`, `l_ub`, and `l_stride` are provided for efficient traversal of local array elements.

ghost_size: Ghost area size in this dimension, both ends. This is intended as a support for shift operations.

p_shape: Number of processors in the processor grid. Note, rank of processor grid may be different from that of array.

local_coord: Coordinate of this processor in the processor grid.

slice_coord: Coordinates of the sub processor grid that holds this distributed array/section. -1 is used for "full dimension".

Per-array information

base_address: Base address of local array. May be different from the address of the first effective element. It may also change upon rank reduced sectioning.

type: Data type code for array element. We define 1 for INTEGER, 2 for REAL, 3 for DOUBLE PRECISION, 4 for CHARACTER.

rank: Rank of the array, changes upon rank reduced array sectioning.

p_rank: Rank of the processor grid.

comm: A handle to the processor grid, for example, MPI_COMM.

major: Majority of array storage, so that the DAD may work for both Fortran and C.

Examples

We show some examples on DAD contents for various situations. In particular, we demonstrate how a DAD gets modified upon sectioning. Algorithms have been designed to fill/modify DAD efficiently.

For the following specification

```
      REAL X(100)
!HPF$ PROCESSORS P(4)
!HPF$ TEMPLATE T(-10:200)
!HPF$ ALIGN X(i) WITH T(2*i-3)
!HPF$ DISTRIBUTE T(BLOCK) ONTO P
```

we would have the following DAD for processor (1)

```
    g_extent :100   0   0   0   0   0   0
    t_extent :212   0   0   0   0   0   0
    t_stride :  2   0   0   0   0   0   0
    t_offset :  9   0   0   0   0   0   0
    distCode : -1   0   0   0   0   0   0
     on_pdim :  1   0   0   0   0   0   0
    l_extent : 53   0   0   0   0   0   0
        l_lb :  9   0   0   0   0   0   0
        l_ub : 51   0   0   0   0   0   0
    l_stride :  2   0   0   0   0   0   0
   ghostSize :  0   0   0   0   0   0   0
     p_shape :  4   0   0   0   0   0   0
    my_coord :  1   0   0   0   0   0   0
  slice_coord : -1   0   0   0   0   0   0
```

Note: (1) the original template is of size 211, and we've brought it up to a multiple of 4, the number of processors; (2) t_offset is 9, since 2*1-3 = -1, and -1 - (-10) = 9; (3) local array is of extent 212/4; (4) assuming local array is 0-based, i.e., X(0:52), the first effective element in processor 1 is X(9).

Now suppose X(4:100:3) is used as an argument to some subroutine. A new DAD, for processor (1), as the following is formed and passed to the subroutine.

```
    g_extent : 33   0   0   0   0   0   0  *
    t_extent :212   0   0   0   0   0   0
    t_stride :  6   0   0   0   0   0   0  *
    t_offset : 15   0   0   0   0   0   0  *
    distCode : -1   0   0   0   0   0   0
     on_pdim :  1   0   0   0   0   0   0
    l_extent : 53   0   0   0   0   0   0
        l_lb : 15   0   0   0   0   0   0  *
        l_ub : 51   0   0   0   0   0   0
    l_stride :  6   0   0   0   0   0   0  *
   ghostSize :  0   0   0   0   0   0   0
     p_shape :  4   0   0   0   0   0   0
    my_coord :  1   0   0   0   0   0   0
  slice_coord : -1   0   0   0   0   0   0
```

Note: (1) changed entries are indicated by *; (2) `l_ub` may also change in general.

For the following specification

```
   INTEGER X(8,6)
(6)PROCESSORS P(4,2)
   ALIGN X(i,j) WITH T(i,j)
   DISTRIBUTE T(BLOCK,BLOCK) ONTO P
   ...
   CALL FOO(X(6,:))
   ...
```

We have dad_X:

```
      g_extent  8   6
      t_extent  8   6
      t_stride  1   1
      t_offset  0   0
     dist_code -1  -1
      on_pdim   1   2
      l_extent  2   3
      l_lb      0   0
      l_ub      1   2
      l_stride  1   1
      ghstsize  0   0
      p_shape   4   2
      my_coord  *   *
   slice_coord -1  -1
                1   2  2  comm  major
                base_address = address of local X(0,0)
```

We include the base_address now, since it will change soon. Now X(6,:) is passed to subroutine, we would have dad_X(6,:),

```
   dad_X(6,:):
      g_extent  6
      t_extent  6
      t_stride  1
      t_offset  0
    dist_code  -2           <- dist_code(2)*l_extent(1)
      on_pdim   2
      l_extent  6           <- l_extent(2)*l_extent(1)
      l_lb      0           <- l_lb(2)*l_extent(1)
      l_ub      4           <- l_ub(2)*l_extent(1)
      l_stride  2           <- l_stride(2)*l_extent(1)
      ghstsize  0
      p_shape   4   2
      my_coord  *   *
   slice_coord  3  -1       changed !
                1   1  2  comm  major
                base_address = address of local X(1,0) !!!
```

Note: the change of base_address is subtle — global X(6,:) is local X(1,:) on the 3rd slice of the two dimensional processor grid.

For the following specification

```
      REAL X(10,20,30)
!HPF$ PROCESSORS P(4,4)
!HPF$ TEMPLATE T(30,50)
!HPF$ ALIGN X(i,*,j) WITH T(j,3*i+5)
!HPF$ DISTRIBUTE T(CYCLIC,CYCLIC) ONTO P
```

we would have the following DAD for processor (1,1)

```
   g_extent : 10  20  30
   t_extent : 52   0  32
   t_stride :  3  -1   1
   t_offset :  7  -1   0
  dist_code :  1   0   1
    on_pdim :  2  -1   1
   l_extent : 13  20   8
       l_lb :  4   0   0
       l_ub :  7  19   7
   l_stride :  3   1   1
 ghost_size :  0   0   0
    p_shape :  4   4
   my_coord :  1   1
slice_coord : -1  -1
            :  2   3   2  comm  major
            :  local base address
```

Now suppose X(:,:,1:30:2) is used as an argument to some subroutine. A new DAD, for processor (1,1), as the following is formed and passed to the subroutine.

```
   g_extent : 10  20  15
   t_extent : 52   0  32
   t_stride :  3  -1   2
   t_offset :  7  -1   0
  dist_code :  1   0   1
    on_pdim :  2  -1   1
   l_extent : 13  20   8
       l_lb :  4   0   0
       l_ub :  7  19   6
   l_stride :  3   1   2
 ghost_size :  0   0   0
    p_shape :  4   4
   my_coord :  1   1
slice_coord : -1  -1
            :  2   3   2  comm  major
            :  local base address, no change
```

Thus, the subroutine effectively sees some array X1(10,20,15) aligned on some template, and distributed on a two dimensional processor grid.

Now suppose this subroutine uses X1(1,:,:) to call another subroutine. We have a rank reduced sectioning situation. We would like the callee to effectively see a two dimensional array. The following DAD would be generated.

```
   g_extent : 20  15
   t_extent :  0  32
   t_stride : -1   2
   t_offset : -1   0
  dist_code :  0   1
    on_pdim : -1   1
   l_extent :260   8
       l_lb :  0   0
       l_ub :247   6
   l_stride : 13   2
 ghost_size :  0   0
    p_shape :  4   4
   my_coord :  1   1
slice_coord :  1  -1
            :  2   2   2  comm  major
            :  local base address, no change.
```

References

[1] HPFF, High Performance Fortran Language Specification (version 1.0). May 3, 1993.

[2] James Cowie, Don Leskiw, and Xiaoming Li, "The Distributed Array Descriptor for a PCRC HPF Compiler," Ver. 1.0, SCCS-770, NPAC, Jan. 31, 1996.

[3] James Cowie, Don Leskiw, and Xiaoming Li, "The Distributed Array Descriptor for a PCRC HPF Compiler," Ver. 1.1, SCCS-770b, NPAC, May 1, 1996.

[4] James Cowie, Don Leskiw, and Xiaoming Li, "The Distributed Array Descriptor for a PCRC HPF Compiler," Ver. 1.2, SCCS-770c, NPAC, Sept 22, 1996.