Appendix A

CM/AVS Routines

This appendix contains descriptions of the supported user-visible routines in the CM/AVS libraries, in alphabetical order.

A module may use the CM/AVS routines to

- allocate parallel arrays
- gain access to the data and coordinates as CM Fortran arrays or C* parallel variables
- query whether or not a field is parallel

CM/AVS provides the following routines:

CMAVScorout_init

CMAVSdata_alloc

CMAVSfield_alloc

CMAVSfield_alloc_data_shape

CMAVSfield_alloc_points_shape

CMAVSfield_copy_points

CMAVSfield_data_get

CMAVSfield_points_get

CMAVSfield_reset_minmax

CMAVSis_field_on_CM

These routines should be used in conjunction with the standard AVS routines. Most of the standard AVS routines also work on parallel fields. The exceptions are listed in Table 1 in Chapter 3.

A.1 CMAVScorout_init

Initializes a CM/AVS coroutine module.

C* Binding

```
#include <cm/cmavs.h>
void
    CMAVScorout_init(int argc, char *argv[],
         int (*desc)());
```

CMF Binding

```
    include '/usr/include/cm/cmavs.inc'
    SUBROUTINE CMAVScorout_init(desc)
    external desc
```

This subroutine should be used instead of **AVScorout_init** to initialize a CM/AVS coroutine module. It must precede any other AVS or CM/AVS routines.

The subroutine sets up some internal data structures, then calls the user-supplied module description function **desc**.

For the C* interface, *argc* and *argv* are the same arguments that are passed to **main**.

A.2 CMAVSdata_alloc

Allocates a parallel field based on a descriptive string.

C* Binding

```
#include <cm/cmavs.h>
void *
    CMAVSdata_alloc(char *string, int *dims);
```

CMF Binding

```
    include '/usr/include/cm/cmavs.inc'
    character*n string
    integer dims()
    integer function CMAVSdata_alloc(string,dims)
```

This routine allocates a parallel field based on a descriptive string. The behavior is similar to **AVSdata_alloc**, except that it allocates the data and points on the CM-5 processing nodes. The **string** argument is a descriptive string in the same form that is used for **AVSdata_alloc**.

NOTE

The **string** argument must describe a field; trying to allocate any other AVS object such as **ucd** or **geom** on the processing nodes will result in a fatal run-time error.

The **dims** argument is an array of integers that tells us how much space to allocate for this field on the CM-5. In C*, this routine returns a pointer to an AVSfield structure. In CM Fortran, it returns an opaque integer that can be used anywhere a parallel field is needed.

A.3 CMAVSfield_alloc

Allocates a field structure for a parallel field using the given template field.

C* Binding

```
#include <cm/cmavs.h>
AVSfield *
    CMAVSfield_alloc(CMAVSfield *template, int *dims)
```

CM Fortran Binding

```
include /usr/include/cm/cmavs.inc'
integer template
integer dims()
integer function CMAVSfield_alloc(template, dims)
```

This routine allocates a field structure for a parallel field using the given template field. The template may be either a parallel or serial AVS field. The newly allocated field will always be a parallel field.

The *dims* argument is an array of integers that tells how much space to allocate for this field on the processing nodes. If you use zero in CM Fortran, or `NULL` in C*, instead of a dimensions array, the dimensions are taken from the template field.

In C*, this routine returns a pointer to an AVSfield structure. In CM Fortran, it returns an opaque integer that can be used anywhere a CM/AVS field is needed.

A.4 CMAVSfield_alloc_data_shape

Allocates a C* shape that can contain the data from a field.

C* Binding

```
#include <cm/cmavs.h>
shape
    CMAVSfield_alloc_data_shape(AVSfield *field)
```

CM Fortran Binding

Not applicable.

This routine allocates a C* shape that can be used to refer to the data in any parallel field having the same rank and dimensions as the intput field. Note that the field's *veclen* does not matter; vectors become C* per-processor arrays that do not affect the choice of shape.

Each time you call **`CMAVSfield_alloc_data_shape`**, a new shape is allocated, even if you use the same field as input.

To deallocate the shape that this routine allocates, you must use the C* routine **`deallocate_shape`**. The field data is not affected when you free the shape that points to it.

A.5 CMAVSfield_alloc_points_shape

Allocate a C* shape that can contain the points from a field.

C* Binding

```
#include <cm/cmavs.h>
shape
    CMAVSfield_alloc_points_shape(AVSfield *field)
```

CM Fortran Binding

Not applicable.

This routine allocates a C* shape that can be used to refer to the coordinates in any parallel field having the same rank and dimensions as the input field.

Do not call this routine on a uniform field; the coordinates for a uniform field can never reside on the processing nodes.

For a rectilinear field, this routine returns a one-dimensional shape with a number of positions equal to the sum of the field dimensions. For an irregular field, it returns a shape of rank *ndim*, where the number of positions in each axis is given by the dimensions array.

Each time you call **`CMAVSfield_alloc_points_shape`**, a new shape is allocated, even if you use the same field as input.

To deallocate the shape that this routine allocates, you must use the C* routine **`deallocate_shape`**. The field coordinates are not affected when you free the shape that points to them.

A.6 CMAVSfield_copy_points

Copies points array from infield to outfield.

C* Binding

```
#include <cm/cmavs.h>
int CMAVSfield_copy_points(AVSfield *infield,
    AVSfield *outfield)
```

CMF Binding

```
include '/usr/include/cm/cmavs.inc'
integer infield, outfield
integer function
         CMAVSfield_copy_points(infield,outfield)
```

This routine copies a points array from an infield to an outfield. It works only if the points arrays are both on the partition manager or both on the processing nodes. The routine returns 1 on success, 0 on failure.

A.7 CMAVSfield_data_get

Gets access to the data portion of a parallel field by returning a pointer to a C* parallel variable or filling in a CM Fortran array descriptor.

C* Binding

```
#include <cm/cmavs.h>
void:void *
    CMAVSfield_data_get(AVSfield *field, shape S)
```

CM Fortran Binding

```
    include '/usr/include/cm/CMF_defs.h'
    include '/usr/include/cm/cmavs.inc'
    integer field
    integer basevec(CMF_SIZEOF_DESCRIPTOR)
CMF$LAYOUT basevec(:serial)
    subroutine CMAVSfield_data_get(field, basevec)
```

This routine returns a pointer to a C* parallel variable or fills in a CM Fortran array descriptor, thereby giving access to the data portion of a parallel field. (The C* parallel variable is allocated in the specified shape.) Once the descriptor is loaded with a CM Fortran array descriptor, it may be passed to any CM Fortran routine that is expecting a parallel array of the appropriate rank.

Note the following:

- The C* interface takes both a field and a shape; it returns a pointer to a parallel variable. The parallel pointer refers to the data in the first argument (**`AVSfield`**). You must pre-allocate the shape using

CMAVSfield_allocate_points_shape. You may use the same shape to construct pointers to any fields that have the same rank and dimensions.

- If you dispose of the field (as with **AVSdata_free**) you should no longer refer to any arrays created from that field.
- A pointer returned by **CMAVS_field_data_get** can be invalidated if you make another call to **CMAVSfield_data_get** on the same field.
- If you use **CMAVSfield_data_get** on a field whose data resides on the partition manager, a fatal error occurs.

A.8 CMAVSfield_points_get

Returns a pointer to the parallel coordinate data from a CM/AVS field.

C* Binding

```
#include <cm/cmavs.h>
float:void *
    CMAVSfield_points_get(AVSfield *field, shape S)
```

CMF Binding

```
    include '/usr/include/cm/CMF_defs.h'
    include '/usr/include/cm/cmavs.inc'
    integer field
    integer basevec(CMF_SIZEOF_DESCRIPTOR)
CMF$LAYOUT basevec(:serial)
    subroutine CMAVSfield_points_get
         (field, basevec)
```

This routine returns a pointer to the parallel coordinate data from a CM/AVS field. It works only when the coordinate array resides on the processing nodes, and it does not work on uniform fields.

Note the following:

- The C* interface takes both a field and a shape; it returns a pointer to a parallel variable. The parallel pointer refers to the data in the first argument (**AVSfield**). You must pre-allocate the shape using **CMAVSfield_allocate_points_shape**. You may use the same shape to construct pointers to any fields that have the same rank and dimensions.

- A pointer returned by **CMAVS_field_points_get** can be invalidated if you make another call to **CMAVSfield_points_get** on the same field.
- If you use **CMAVSfield_points_get** on a field whose data resides on the partition manager, a fatal error occurs.

A.9 CMAVSfield_reset_minmax

Recomputes the minimim and maximum values for the field's computational data and stores those values with the field.

C* Binding

```
#include <cm/cmavs.h>
void
    CMAVSfield_reset_minmax(AVSfield *field)
```

CM Fortran Binding

```
    include '/usr/include/cm/cmavs.inc'
    integer field
    SUBROUTINE CMAVSfield_reset_minmax(field)
```

This routine recomputes the **min** and **max** values for the field's computational data and stores those values with the field. The routine works for both parallel and serial fields.

A.10 CMAVSis_field_on_CM

Accepts a pointer to an AVS field, and returns true if the field is a parallel field.

C* Binding

```
#include <cm/cmavs.h>
bool
    CMAVSis_field_on_CM(AVSfield *field);
```

CM Fortran Binding

```
include '/usr/include/cm/cmavs.inc'
integer field
logical function CMAVSis_field_on_CM(field)
```