Chapter 6

Dynamic Array Allocation

Dynamic allocation of storage is supported through the ALLOCATE and DEALLOCATE statements, which manage the storage of allocatable arrays and pointers.

Allocatable arrays are declared with the attribute ALLOCATABLE; pointers are arrays declared with the attribute POINTER. The size, bounds, and shape of an allocatable array or pointer are undefined until the array is allocated by the execution of an ALLOCATE statement. They can then be used like any other CM array. They cannot, however, be function results.

Pointers that are local to a program unit provide the same capabilities as allocatable arrays. The difference is that pointers can appear in common, whereas allocatable arrays cannot. Pointers thus provide the ability to manage global memory dynamically.

6.1 Declaring Allocatable Arrays and Pointers

Allocatable arrays are declared either with a type declaration statement that specifies the ALLOCATABLE attribute (see Section 5.4.6) or by the ALLOCATABLE statement (see Section 5.5.5):

```
REAL, ALLOCATABLE, DIMENSION(:) :: A,B
```

or,

```
...
ALLOCATABLE A,B
```


Copyright © 1989–1994 Thinking Machines Corporation

Pointers are declared either with a type declaration statement that specifies the POINTER attribute (see Section 5.4.7) or by the POINTER statement (see Section 5.5.6):

```
REAL, POINTER, DIMENSION(:) :: C,D
```

or,

```
...
POINTER C,D
```

The allocatable array or pointer is defined with a *deferred-shape-array-spec* (R5-8), in which colons are used to describe the rank of the array. For example, the following declaration declares a two-dimensional allocatable array:

```
REAL, ALLOCATABLE :: A(:,:)
```

The extents of the array dimensions are specified when the array is allocated by the ALLOCATE statement.

The deferred-shape specification can be attached to the array name in a type declaration or POINTER or ALLOCATABLE statement, or it can be supplied with a separate DIMENSION specification. For example, here are several equivalent ways to declare three pointers, P, Q, and R.

```
INTEGER, POINTER :: P(:), Q(:), R(:,:)
```

or,

```
INTEGER P,Q,R
POINTER P(:), Q(:), R(:,:)
```

or,

```
INTEGER, POINTER, DIMENSION(:) :: P,Q
INTEGER, POINTER, DIMENSION(:,:) :: R
```

or,

```
INTEGER P,Q,R
DIMENSION P(:), Q(:), R(:,:)
POINTER P,Q,R
```


Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

6.2 Allocation Status

At the beginning of program execution, the allocation status of an allocatable array or pointer is *not currently allocated*, and the array or pointer cannot be defined. After the successful execution of an ALLOCATE statement, the array or pointer is *allocated* and is available for use. It is an error to use ALLOCATE on an array that is already allocated, but it is not an error to use ALLOCATE on a pointer that is already allocated.

When the DEALLOCATE statement is used for an allocated array or pointer, its status becomes *not currently allocated*, and it is again undefinable. It is an error to use the DEALLOCATE statement for an array or pointer that is not currently allocated.

If an allocated array or a local pointer is used in a procedure, its allocation status becomes *undefined* when the procedure ends. An array or pointer with an undefined allocation status cannot be referenced, defined, allocated, or deallocated. For more information about pointers as arguments, see Sections 12.6.3.2 and 12.6.3.3.

To determine the allocation status of an allocatable array, use the intrinsic function ALLOCATED (see Chapter 17).

To determine the allocation status of a pointer, use the intrinsic function ASSOCIATED (see Chapter 17).

```
ASSOCIATED( POINTER )
```

For example:

```
IF (ASSOCIATED(P)) THEN
    PRINT *, 'P is already allocated.'
ELSE
    ALLOCATE (P(0:M))
END IF
```

6.3 ALLOCATE Statement

The ALLOCATE statement dynamically allocates storage in memory for allocatable arrays and pointers.

An *allocate-stmt* is: R6-1

ALLOCATE (*allocation(s)* [, STAT= *stat-variable*])

An *allocation* is: R6-2

allocate-object (*explicit-shape-spec(s)*)

Copyright © 1989–1994 Thinking Machines Corporation

An *allocate-object* is: R6-3
variable-name

An *explicit-shape-spec* is: R5-3
[lower-bound **:** *] upper-bound*

A *stat-variable* is: R6-4
scalar-int-variable

- Each *allocate-object* must be the name of an ALLOCATABLE array that is not currently allocated or a POINTER.
- A bound in an *allocation explicit-shape-spec* must not be an expression containing an array inquiry function whose argument is any other object in the same ALLOCATE statement.
- The number of *explicit-shape-spec*s in an *allocation* must be the same as the rank of the deferred-shape array.
- The *stat-variable* must not be allocated within the ALLOCATE statement in which it appears.

An example of an ALLOCATE statement is:

```
ALLOCATE ( X(N), B(-3:M, 0:9), STAT=STAT_VAR )
```

The values of the lower bound and upper bound expressions in an *explicit-shape-spec* when an ALLOCATE statement is executed determine the bounds of the array. Subsequent redefinition or undefinition of any entities in the bound expressions do not affect the array shape. If the lower bound is not specified, it is assumed to be 1.

If the STAT= specifier is present, it is defined with the value of zero upon successful execution of the ALLOCATE statement; any positive value indicates failure. If the allocation operation fails in the absence of a STAT= specifier, program execution terminates.

It is an error to allocate an allocatable array that is currently allocated. However, it is not an error to allocate a pointer that is already allocated; in this case, the pointer becomes associated with the new storage, and the previous storage becomes inaccessible.

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

6.4 DEALLOCATE Statement

The DEALLOCATE statement causes an allocatable array or a pointer to be deallocated and the pointer to be disassociated.

A *deallocate-stmt* is: R6-5
DEALLOCATE (*allocate-object(s) [* , STAT= *stat-variable]*)

An *allocate-object* is: R6-3
variable-name

- Each *allocate-object* must be an ALLOCATABLE array that is currently allocated or an array with the attribute POINTER that is currently allocated.
- The *stat-variable* must not be deallocated within the DEALLOCATE statement in which it appears.

When an array is deallocated its allocation status becomes *not currently allocated*. Deallocating an array that is not currently allocated causes an error condition in the DEALLOCATE statement.

If the STAT= specifier is present, it is defined with the value of zero upon successful execution of the DEALLOCATE statement; any positive value indicates failure. If the deallocation operation fails in the absence of a STAT= specifier, program execution terminates.

An example of a DEALLOCATE statement:

```
DEALLOCATE (X,B, STAT=STAT_VAR)
```


Copyright © 1989–1994 Thinking Machines Corporation