Chapter 2

Inquiries

2.1 System Inquiry Functions

Three functions report information about the CM system that is executing the program. They all take no arguments and return integer scalar results.

CMF_ARCHITECTURE returns a predefined constant that identifies the CM hardware platform and execution model:

```
ARCH = CMF_ARCHITECTURE( )
```

CMF_NUMBER_OF_PROCESSORS reports the number of processing elements available:

```
NUM = CMF_NUMBER_OF_PROCESSORS( )
```

The table below shows the return values of these two inquiry functions. Notice that the CM system component that serves as the "processing element" is different for the various platforms and execution models.

Copyright © 1994 Thinking Machines Corporation

Table 1. CM Fortran hardware platforms and execution models.

	Compiler options	CMF_ARCHITECTURE returns	CMF_NUMBER_OF_PROCESSORS returns
CM-5			
Vector units	**-cm5 -vu**	**CMF_CM5_VU**	number of vector units
SPARC nodes	**-cm5 -sparc**	**CMF_CM5_SPARC**	number of nodes
CM-200			
Slicewise	**-cm200 -slicewise**	**CMF_CM200_SLICEWISE**	number of nodes
CM-2			
Slicewise	**-cm2 -slicewise**	**CMF_CM2_SLICEWISE**	number of nodes
CM Fortran Simulator	**-cmsim**	**CMF_CMSIM**	number of processors (1)

See the *CM Fortran User's Guide* for more information on execution models and the hardware platforms they support.

A third inquiry function, **CMF_AVAILABLE_MEMORY**, reports the number of bytes of memory still available in each processing element:

```
BYTES = CMF_AVAILABLE_MEMORY( )
```

NOTE: **CMF_ARCHITECTURE** and **CMF_AVAILABLE_MEMORY** return the same values for the nodal model and the local part of the global/local model (specified by **-cm5 -vu -node ...** and **-cm5 -vu ... -local**, respectively) as they do for the vector-units model. **CMF_NUMBER_OF_PROCESSORS** returns the number of vector units on a node (4).

2.1.1 Language Comparison

No comparable language feature.

Version 2.1, January 1994
Copyright © 1994 Thinking Machines Corporation

2.1.2 An Additional System Inquiry Function

The inquiry function **CMF_ARRAY_SIZE_QUANTUM** is described in Section 5.5, in the context of array aliasing.

Copyright © 1994 Thinking Machines Corporation

CMF_ARCHITECTURE

Identifies the CM hardware model and execution model under which the program is executing.

Syntax

```
INCLUDE '/usr/include/cm/CMF_defs.h'

ARCH = CMF_ARCHITECTURE( )
```

Arguments

None.

Result

An integer constant, one of the following:

CMF_CM5_VU

CMF_CM5_SPARC

CMF_CM200_SLICEWISE

CMF_CM2_SLICEWISE

CMF_SIM

Description

This function returns a constant that identifies the CM model (CM-5, CM-2, or CM-200) and the execution model under which a program is running. On the CM-2 and CM-200 the execution model is slicewise. On the CM-5 the execution model can be SPARC (indicating a CM-5 that is not using vector units) or vector units (indicating a CM-5 that is using vector units in addition to the SPARC processors). Finally, **CMF_SIM** indicates that the program is running on a Sun computer under the CM Fortran simulator.

Version 2.1, January 1994
Copyright © 1994 Thinking Machines Corporation

Note

CMF_ARCHITECTURE returns **CMF_CM5_VU** for the nodal model and the local part of the global/local model (specified by **-cm5 -vu -node ...** and **-cm5 -vu ... -local**, respectively).

See Also

CMF_NUMBER_OF_PROCESSORS

Copyright © 1994 Thinking Machines Corporation

CMF_NUMBER_OF_PROCESSORS

Returns the number of parallel processing elements available to the program in the current execution environment.

Syntax

```
INCLUDE '/usr/include/cm/CMF_defs.h'

NUMBER = CMF_NUMBER_OF_PROCESSORS( )
```

Arguments

None.

Result

Integer scalar.

Description

The hardware component considered a processing element by **CMF_NUMBER_OF_PROCESSORS** varies with the CM architecture and the execution model under which the program is running. The machine and execution model can be determined with the CM Fortran utility **CMF_ARCHITECTURE**. The following table summarizes the meaning of the return value of **CMF_NUMBER_OF_PROCESSORS** for each value returned by **CMF_ARCHITECTURE**:

Version 2.1, January 1994
Copyright © 1994 Thinking Machines Corporation

CMF_ARCHITECTURE Return Values	**CMF_NUMBER_OF_PROCESSORS** Return Values
CMF_CM5_SPARC	number of processing nodes
CMF_CM5_VU	number of vector units
CMF_CM200_SLICEWISE	number of processing nodes
CMF_CM2_SLICEWISE	number of processing nodes
CMF_SIM	number of processors (1)

Note

CMF_NUMBER_OF_PROCESSORS returns the number of vector units on a node for the nodal model and the local part of the global/local model (specified by **-cm5 -vu -node ...** and **-cm5 -vu ... -local**, respectively).

See Also

CMF_ARCHITECTURE

Copyright © 1994 Thinking Machines Corporation

CMF_AVAILABLE_MEMORY

Returns the number of bytes available in each parallel processing element.

Syntax

```
INCLUDE '/usr/include/cm/CMF_defs.h'

MEMORY = CMF_AVAILABLE_MEMORY( )
```

Result

Integer scalar.

Description

This function returns an integer reporting, in units of bytes, the amount of memory left in each processing element available to the program. The hardware component that serves as the processing element differs according to the CM model and execution model being used (as returned by **CMF_ARCHITECTURE**). The significance of the available memory values is shown in the following table.

CMF_ARCHITECTURE Return Values	**CMF_AVAILABLE_MEMORY** Return Values
CMF_CM5_SPARC	memory remaining per processing node
CMF_CM5_VU	memory remaining per vector unit
CMF_CM200_SLICEWISE	memory remaining per processing node
CMF_CM2_SLICEWISE	memory remaining per processing node
CMF_SIM	memory remaining in the data structures used to simulate distributed memory

Copyright © 1994 Thinking Machines Corporation

Note

CMF_AVAILABLE_MEMORY returns **CMF_CM5_VU** for the nodal model and the local part of the global/local model (specified by **-cm5 -vu -node ...** and **-cm5 -vu ... -local**, respectively).

Copyright © 1994 Thinking Machines Corporation

2.2 Array Inquiry Subroutine

2.2.1 Array Shape and Layout

The subroutine **CMF_DESCRIBE_ARRAY** prints information about a CM array to standard output:

```
CALL CMF_DESCRIBE_ARRAY( ARRAY )
```

The output includes the home, rank, and dimension extents of the array, as well as detailed information about its layout on the processing elements. For example, for an array declared as:

```
      DOUBLE PRECISION, ARRAY (128,128,8,16) :: B
CMF$  LAYOUT B(:SERIAL,:SERIAL,:NEWS,:NEWS)
```

CMF_DESCRIBE_ARRAY gives the following output when running on a 4-node (16 vector unit) partition:

```
Array geometry id: 0xa3610
    Rank: 4
    Number of elements: 2097152
    Extents: [128 128 8 16]
    Machine geometry id: 0xa9e50, rank: 4
    Machine geometry elements:  2097152
    Overall subgrid size:       131072
    Axis 0 (serial axis):
         Extent:    128 (1 physical x 128 subgrid)
         Off-chip:  0 bits, mask = 0x0
         Subgrid:   length = 128, axis-increment = 1024
    Axis 1 (serial axis):
         Extent:    128 (1 physical x 128 subgrid)
         Off-chip:  0 bits, mask = 0x0
         Subgrid:   length = 128, axis-increment = 8
    Axis 2:
         Extent:    8 (2 physical x 4 subgrid)
         Off-chip:  1 bits, mask = 0x8
         Subgrid:   length = 4, axis-increment = 2
    Axis 3:
         Extent:    16 (8 physical x 2 subgrid)
         Off-chip:  3 bits, mask = 0x7
         Subgrid:   length = 2, axis-increment = 1
```

Notice that the two **:serial** axes are laid out entirely on-processor: their physical extents on the virtual grid are both 1, there are no “off-chip” (mean-

Version 2.1, January 1994
Copyright © 1994 Thinking Machines Corporation

ing "off-processor") bits for those axes in the array's send address, and the subgrid length equals the declared axis length. The two **:NEWS** axes are distributed across 2 and 8 physical processors, respectively, as indicated by their physical extents and their bitmasks. The rightmost **:NEWS** axis is contiguous in local memory (axis-increment = 1). This array is not padded, since the product of its **:NEWS** axes is a legal size for a virtual grid in the run-time environment (16 vector units).

2.2.2 Array Descriptor Inquiries

The following procedures derive information from an array descriptor. They are described in Section 5.4 in the context of array aliasing.

```
INT = CMF_GET_GLOBAL_LOCATION( ARRAY, OFFSETS, ASTAT )

INT = CMF_GET_PHYSICAL_LOCATION( ARRAY, OFFSETS, ASTAT )

CMF_GET_AXIS_ORDERS( ARRAY, ORDERS, RANK, ASTAT )

CMF_GET_LOCAL_EXTENTS( ARRAY, EXTENTS, RANK, ASTAT )

CMF_GET_LOCAL_STRIDES( ARRAY, STRIDES, RANK, ASTAT )

CMF_GET_PROCESSOR_MASKS( ARRAY, MASKS, RANK, ASTAT )
```

2.2.3 Special-Purpose Array Inquiries

The Utility Library also provides two special-purpose array inquiry functions.

- **CMF_GET_GEOMETRY_ID** is used only in constructing destination addresses for scatter operations; it is described in Section 6.2.2.
- **CMF_SIZEOF_ARRAY_ELEMENT** is used only for certain operations on CM files; it is described in Section 7.3.

2.2.4 Language Comparison

No comparable language features.

Copyright © 1994 Thinking Machines Corporation

CMF_DESCRIBE_ARRAY

Prints information about a CM array to **stdout**, including its shape and detailed information about its layout.

Syntax

```
INCLUDE '/usr/include/cm/CMF_defs.h'

CALL CMF_DESCRIBE_ARRAY( ARRAY )
```

Arguments

ARRAY	A CM array of any type.

Result

None.

Version 2.1, January 1994
Copyright © 1994 Thinking Machines Corporation