Part II: Detailed Information about New Functions and Subroutines

10 New Fortran 90 Intrinsic Functions

This section contains man pages for the new Fortran 90 intrinsic functions described in Section 4.1.1.

Copyright © 1994 Thinking Machines Corporation

BIT_SIZE

Returns the number of bits in the model representing numbers of the same kind type as the argument.

Syntax

```
BIT_SIZE( I )
```

Kind

Inquiry function.

Argument

I	A scalar or array of type **INTEGER**.

Result Type

Scalar integer of the same kind type as **I**.

Result Value

The result is one of the following values, depending on the kind type of the argument.

Data Type	Value
INTEGER*4	32
INTEGER*8	64_64

Note

The **INTEGER*8** type is supported only on CM-5 systems with vector units.

Version 2.2, October 1994
Copyright © 1994 Thinking Machines Corporation

CEILING

Returns the least integer greater than or equal to **A**.

Syntax

```
CEILING( A )
```

Kind

Elemental function.

Argument

A	A scalar or array of type **REAL**.

Result Type

A scalar or array of type **INTEGER*4**.

Result Value

The result is equal to the least integer greater than or equal to **A**. The result is undefined if the processor cannot represent this value as an **INTEGER*4**.

Examples

```
CEILING(3.7) → 4

CEILING(-3.7) → -3
```

Copyright © 1994 Thinking Machines Corporation

DATE_AND_TIME

Returns integer data from the date and real-time clock.

Syntax

```
DATE_AND_TIME([ DATE ][ ,TIME ][ ,ZONE] [ ,VALUES ])
```

Kind

Subroutine

Arguments

DATE (optional) A scalar of type **CHARACTER**, with a length of at least 8, in order to contain the complete value. It is an **INTENT(OUT)** argument. Its leftmost 8 characters are set to a value of the form CCYY MMDD, where CC is the century, YY the year within the century, MM the month within the year, and DD the day within the month. If there is no data available, **DATE** is blank.

TIME (optional) A scalar of type **CHARACTER**, with a length of at least 10 in order to contain the complete value. It is an **INTENT(OUT)** argument. Its leftmost 10 characters are set to a value of the form HHMMSS.SSS, where HH is the hour of the day, MM is the minutes of the hour, and SS.SSS is the seconds and milliseconds of the minute. If there is no clock available, **TIME** is blank.

ZONE (optional) A scalar of type **CHARACTER**, with a length of at least 5 in order to contain the complete value. It is an **INTENT(OUT)** argument. Its leftmost 5 characters are set to a value of the form +/−HHMM, where HH and MM are the time difference with respect to Coordinated Universal Time (UTC), also known as Greenwich Mean Time, in hours and minutes, respectively. If there is no clock available, **ZONE** is blank.

Copyright © 1994 Thinking Machines Corporation

VALUES (optional)

Rank one **INTEGER*4**. It is an **INTENT(OUT)** argument. Its size must be at least 8. The values returned in **VALUES** are as follows:

VALUES(1) the year (for example, 1994)
VALUES(2) the month of the year
VALUES(3) the day of the month
VALUES(4) the time difference with respect to Coordinated Universal Time (UTC) in minutes
VALUES(5) the hour of the day, in the range of 0 to 23
VALUES(6) the minutes of the hour, in the range 0 to 59
VALUES(7) the seconds of the minute, in the range 0 to 59
VALUES(8) the milliseconds of the second, in the range 0 to 999

Example

```
 INTEGER DATE-TIME (8)
 CHARACTER (LEN=10) BIG-BEN (3)
 CALL DATE_AND_TIME (BIG-BEN(1), BIG-BEN(2), BIG-BEN(3),
&                DATE-TIME)
```

If called in Geneva, Switzerland on 1985 April 12 at 15:27:35.5, this code would assign the value 19850412 to **BIG-BEN(1)**, the value 152735.500 to **BIG-BEN(2)**, and the value +0100 to **BIG-BEN(3)**, and the following values to **DATE-TIME**: 1985, 4, 12, 60, 15, 27, 35, 500.

Copyright © 1994 Thinking Machines Corporation

DIGITS

Returns the number of significant digits used to represent numbers of the same type and kind type as the argument.

Syntax

```
DIGITS( X )
```

Kind

Inquiry function.

Argument

X A scalar or array of type **INTEGER** or **REAL**.

Result Type

Scalar of type **INTEGER*4**.

Result Value

The result is one of the following values, depending on the kind type of the argument.

Data Type	Value
INTEGER*4	31
INTEGER*8	63
REAL*4	24
REAL*8	53

Version 2.2, October 1994
Copyright © 1994 Thinking Machines Corporation

Note

The **`INTEGER*8`** type is supported only on CM-5 systems with vector units.

Copyright © 1994 Thinking Machines Corporation

EPSILON

Returns a positive number that is almost negligible compared to unity in the model representing numbers of the same type and kind type as the argument.

Syntax

```
EPSILON( X )
```

Kind

Inquiry function.

Argument

X A scalar or array of type **REAL**.

Result Type

Scalar of the same type and kind type as **X**.

Result Value

The result has the value 2^{1-p} where p is as defined in Section 4.1.2 for the model representing numbers of the same type and kind type as **X**.

The result is one of the following values, depending on the kind type of the argument.

Data Type	Value
REAL*4	2^{-23}
REAL*8	2^{-52}

Copyright © 1994 Thinking Machines Corporation

EXPONENT

Returns the exponent part of the argument when it is represented as a model number.

Syntax

```
EXPONENT( X )
```

Kind

Elemental function.

Argument

X	A scalar or array of type **REAL**.

Result Type

Scalar of type **INTEGER*4**.

Result Value

The result has a value equal to the exponent *e* of the model representation defined in Section 4.1.2 for the value of X, provided **X** is nonzero and *e* is within the range for integers. **EXPONENT(X)** has the value zero if **X** is zero.

In other words, **EXPONENT(X)** is the least power to which 2 is raised, to yield a value greater than or equal to **X**.

Examples

```
EXPONENT(1.0) → 1

EXPONENT(4.1) → 3
```

Copyright © 1994 Thinking Machines Corporation

FLOOR

Returns the greatest integer less than or equal to **A**.

Syntax

```
FLOOR( A )
```

Kind

Elemental function.

Argument

A	A scalar or array of type **REAL**.

Result Type

A scalar or array of type **INTEGER*4**.

Result Value

The result is equal to the greatest integer less than or equal to **A**. The result is undefined if the processor cannot represent this value as an **INTEGER*4**.

Examples

```
FLOOR(3.7)  → 3

FLOOR(-3.7)  → -4
```

Copyright © 1994 Thinking Machines Corporation

FRACTION

Returns the fractional part of the argument when it is represented as a model number.

Syntax

```
FRACTION( X )
```

Kind

Elemental function.

Argument

X	A scalar or array of type **REAL**.

Result Type

Same as **X**.

Result Value

The result has the value $\mathbf{X} \times 2^{-e}$, where e is as defined in Section 4.1.2 for the model representation of **X**. If **X** has the value zero, the result has the value zero.

FRACTION(X) $\times$ $2^{\text{EXPONENT(X)}}$ = X.

Example

```
FRACTION(3.0)  →  0.75
```

Copyright © 1994 Thinking Machines Corporation

HUGE

Returns the largest number in the model representing numbers of the same type and kind type as the argument.

Syntax

```
HUGE( X )
```

Kind

Inquiry function.

Argument

X	A scalar or array of type **INTEGER** or **REAL**.

Result Type

Scalar of the same type and kind type as **X**.

Result Value

The result has the value 2^q-1 if **X** is of type integer and $(1-2^{-p})2^{e}_{\max}$ if **X** is of type real, where q, p, and $e_{\max}$ are as defined in Section 4.1.2 for the model representing numbers of the same type and kind type as **X**.

The result is one of the following values, depending on the kind type of the argument.

Data Type	Value
INTEGER*4	$2^{31}-1$
INTEGER*8	$2^{63}-1$
REAL*4	$2^{128}(1-2^{-24})$
REAL*8	$2^{1024}(1-2^{-53})$

Copyright © 1994 Thinking Machines Corporation

MAXEXPONENT

Returns the maximum exponent in the model representing numbers of the same type and kind type as the argument.

Syntax

```
MAXEXPONENT( X )
```

Kind

Inquiry function.

Argument

X A scalar or array of type **REAL**.

Result Type

Scalar of type **INTEGER*4**.

Result Value

The result has the value e_{max}, as defined in Section 4.1.2 for the model representing numbers of the same type and kind type as **X**.

The result is one of the following values, depending on the kind type of the argument.

Data Type	Value
REAL*4	128
REAL*8	1024

Copyright © 1994 Thinking Machines Corporation

MINEXPONENT

Returns the minimum (most negative) exponent in the model representing numbers of the same type and kind type as the argument.

Syntax

```
MINEXPONENT( X )
```

Kind

Inquiry function.

Argument

X A scalar or array of type **REAL**.

Result Type

Scalar of type **INTEGER*4**.

Result Value

The result has the value e_{min}, as defined in Section 4.1.2 for the model representing numbers of the same type and kind type as **X**.

The result is one of the following values, depending on the kind type of the argument.

Data Type	Value
REAL*4	−125
REAL*8	−1021

Version 2.2, October 1994
Copyright © 1994 Thinking Machines Corporation

MODULO

Modulo function.

Syntax

```
MODULO( A, P )
```

Kind

Elemental function.

Arguments

A	A scalar or array of type **INTEGER** or **REAL**.
P	Must be of the same type and kind type as **A**.

Result Type

Same as **A**.

Result Value

Case 1 **A** is of type integer. If **P** is not equal to 0, the value of the result is **A - FLOOR(REAL(A)/REAL(P)) * P**. If **P** equals 0, the result is processor-defined.

Case 2 **A** is of type real. If **P** is not equal to 0, the result is **A - FLOOR(A/P) * P**. If **P** equals 0, the result is processor-defined.

Examples

```
MODULO(8, 5) → 3
MODULO(-8, 5) → 2
MODULO(8, -5) → -2
MODULO(-8, -5) → -3
```

Copyright © 1994 Thinking Machines Corporation

NEAREST

Returns the nearest different processor-representable number in a given direction.

Syntax

```
NEAREST( X, S )
```

Kind

Elemental function.

Arguments

X	A scalar or array of type **REAL**.
S	A scalar or array of type **REAL**. Must not equal zero.

Result Type

Same as **X**.

Result Value

The result has a value equal to the processor-representable number distinct from **X** and nearest to it in the direction of the infinity with the same sign as **S**.

Example

If **X** is represented by the model described in Section 4.1.2, then

NEAREST(3.0, 2.0) $\rightarrow$ $3+2^{-22}$

Copyright © 1994 Thinking Machines Corporation

PRECISION

Returns the decimal precision in the model representing real numbers of the same kind type as the argument.

Syntax

```
PRECISION( X )
```

Kind

Inquiry function.

Argument

X	A scalar or array of type **REAL** or **COMPLEX**.

Result Type

Scalar of type **INTEGER*4**.

Result Value

The result has the value INT(LOG10(2^{p-1})), where p is as defined in Section 4.1.2 for the model representing real numbers of the same kind type as **X**.

The result is one of the following values, depending on the kind type of the argument:

Data Type	Value
REAL*4	6
REAL*8	15
COMPLEX*8	6
COMPLEX*16	15

Copyright © 1994 Thinking Machines Corporation

RADIX

Returns the base of the model representing numbers of the same type and kind type as the argument.

Syntax

```
RADIX( X )
```

Kind

Inquiry function.

Argument

X A scalar or array of type **INTEGER** or **REAL**.

Result Type

Scalar of type **INTEGER*4**.

Result Value

The result is one of the following values, depending on the kind type of the argument:

Data Type	Value
INTEGER*4	2
INTEGER*8	2
REAL*4	2
REAL*8	2

Version 2.2, October 1994
Copyright © 1994 Thinking Machines Corporation

Note

The **`INTEGER*8`** type is supported only on CM-5 systems with vector units.

Copyright © 1994 Thinking Machines Corporation

RANDOM_NUMBER

Returns one pseudo random number or an array of pseudo random numbers from the uniform distribution over the range **0 <= x < 1**.

Syntax

RANDOM_NUMBER(HARVEST)

Kind

Subroutine.

Argument

HARVEST	A scalar or CM array of type **REAL** or **DOUBLE PRECISION**. It is an **INTENT(OUT)** argument.

Examples

```
REAL X, Y (10, 10)
CALL RANDOM_NUMBER (HARVEST = X)
! INITIALIZES X WITH A PSEUDO RANDOM NUMBER
CALL RANDOM_NUMBER (Y)
! X AND Y CONTAIN UNIFORMLY DISTRIBUTED RANDOM NUMBERS
```

Note

The performance of this subroutine is significantly improved if the program is linked with the CM Scientific Software Library, **-lcmssl**.

Version 2.2, October 1994
Copyright © 1994 Thinking Machines Corporation

RANDOM_SEED

Restarts or queries the pseudo random number generator.

Syntax

```
RANDOM_SEED([SIZE] [,PUT] [,GET])
```

Kind

Subroutine.

Arguments

There must either be exactly one or no arguments present.

SIZE (optional) The number of integers used to hold the value of the seed. **SIZE** is always 1.

PUT (optional) **INTEGER*4** array of rank one. Sets the seed value for both the CM and front-end generators.

GET (optional) **INTEGER*4** front-end array of rank one, set to the current value of the seed.

If no argument is present, the processor sets the seed to a processor-dependent value.

Copyright © 1994 Thinking Machines Corporation

RANGE

Returns the decimal exponent range in the model representing integer or real numbers of the same kind type as the argument.

Syntax

```
RANGE( X )
```

Kind

Inquiry function.

Argument

X	A scalar or array of type **INTEGER**, **REAL**, or **COMPLEX**.

Result Type

Scalar of type **INTEGER*4**.

Result Value

Case 1	For an integer argument, the result has the value **INT** (**LOG10** (*huge*)), where *huge* is the largest positive integer in the model representing integer numbers of the same kind type as **X** (Section 4.1.2).
Case 2	For a real or complex argument, the result has the value **INT** (**MIN** (**LOG10** (*huge*) − **LOG10** (*tiny*))), where *huge* and *tiny* are the largest and smallest positive numbers in the model representing real numbers of the same kind type as **X** (Section 4.1.2).

Copyright © 1994 Thinking Machines Corporation

The result is one of the following values, depending on the kind type of the argument:

Data Type	Value
`INTEGER*4`	9
`INTEGER*8`	18
`REAL*4`	37
`REAL*8`	307
`COMPLEX*8`	37
`COMPLEX*16`	307

Note

The **`INTEGER*8`** type is supported only on CM-5 systems with vector units.

Copyright © 1994 Thinking Machines Corporation

RRSPACING

Returns the reciprocal of the relative spacing of model numbers near the argument value.

Syntax

```
RRSPACING( X )
```

Kind

Elemental function.

Argument

X	A scalar or array of type **REAL**.

Result Type

Same as **X**.

Result Value

The result has the value $|\mathbf{X} \times 2^{-e}| \times 2^{p}$, where e and p are as defined in Section 4.1.2 for the model representation of **X**, provided this result is within range.

Spacing is the same for all numbers with a given exponent. Since those numbers can differ greatly in size, **RRSPACING** provides information about the relative spacing.

Example

If **X** is represented by the model described in Section 4.1.2, then

RRSPACING(-3.0) $\rightarrow$ .75 x 2^{24}

Version 2.2, October 1994
Copyright © 1994 Thinking Machines Corporation

SCALE

Multiplies a real times its base to an integer power.

Syntax

```
SCALE( X, I )
```

Kind

Elemental function.

Arguments

X	A scalar or array of type **REAL**.
I	A scalar or array of type **INTEGER**, conformable to **X**.

Result Type

Same as **X**.

Result Value

The result has the value **X** x 2^{I}, provided this result is within range.

Example

```
SCALE(3.0, 2) → 12.0
```


Copyright © 1994 Thinking Machines Corporation

SELECTED_INT_KIND

Returns the smallest value of the kind type parameter of an integer data type that represents all integer values n with $-10^R < n < 10^R$.

Syntax

```
SELECTED_INT_KIND( R )
```

Kind

Transformational function.

Argument

R	A scalar of type **INTEGER**.

Result Type

Scalar of type **INTEGER*4**.

Result Value

The result has a value equal to the smallest value of the kind type parameter of an integer data type that represents all values n in the range of values n, with $-10^R < n < 10^R$ or, if no such kind type is available on the processor, the result is -1.

Copyright © 1994 Thinking Machines Corporation

SELECTED_REAL_KIND

Returns the smallest value of the kind type parameter of a real data type with decimal precision of at least **P** digits and a decimal exponent range of at least **R**.

Syntax

```
SELECTED_REAL_KIND([ P ] [,R ])
```

Kind

Transformational function.

Arguments

P (optional) A scalar of type **INTEGER**.

R (optional) A scalar of type **INTEGER**.

Result Type

Scalar of type **INTEGER*4**.

Result Value

The result has a value equal to the smallest value of the kind type parameter of a real data type with decimal precision, as returned by the function **PRECISION**, of at least **P** digits and a decimal exponent range, as returned by the function **RANGE**, of at least **R**. If no such kind type is available on the processor, the result is −1 if the precision is not available, −2 if the exponent range is not available, and −3 if neither is available.

Copyright © 1994 Thinking Machines Corporation

Note

At least one argument must be present.

Version 2.2, October 1994
Copyright © 1994 Thinking Machines Corporation

SET_EXPONENT

Returns the model number whose fractional part is the fractional part of the model representation of **X** and whose exponent part is **I**.

Syntax

```
SET_EXPONENT( X, I )
```

Kind

Elemental function.

Arguments

X	A scalar or array of type **REAL**.
I	A scalar or array of type **INTEGER**, conformable to **X**.

Result Type

Same as **X**.

Result Value

The result has the value **X** x 2^{I-e}, where e is as defined in Section 4.1.2 for the model representation of **X**, provided this result is within range. If **X** is zero, the result is zero.

Example

If **X** is represented by the model described in Section 4.1.2, then

```
SET_EXPONENT(3.0, 1) → 1.5
```

Copyright © 1994 Thinking Machines Corporation

SPACING

Returns the absolute spacing of model numbers near the argument value.

Syntax

```
SPACING( X )
```

Kind

Elemental function.

Argument

X	A scalar or array of type **REAL**.

Result Type

Same as **X**.

Result Value

The result has the value 2^{e-p}, where e and p are as defined in Section 4.1.2 for the model representation of **X**, provided this result is within range. Otherwise, the result is the same as that of **TINY(X)**.

The result is the representable difference between two successive numbers near **X**. This spacing is constant for all arguments with a given exponent.

Example

If **X** is represented by the model described in Section 4.1.2, then

SPACING(3.0) $\rightarrow$ 2^{-22}

Copyright © 1994 Thinking Machines Corporation

SYSTEM_CLOCK

Returns integer data from a real-time clock.

Syntax

```
SYSTEM_CLOCK([ COUNT ][, COUNT_RATE ][, COUNT_MAX ])
```

Kind

Subroutine.

Arguments

COUNT (optional)
: A scalar of type **INTEGER*4**. It is an **INTENT(OUT)** argument. The number of milliseconds since an arbitrary zero point.

COUNT_RATE (optional)
: A scalar of type **INTEGER*4**. It is an **INTENT(OUT)** argument. **COUNT_RATE** is always 1000.

COUNT_MAX (optional)
: A scalar of type **INTEGER*4**. It is an **INTENT(OUT)** argument. **COUNT_MAX** is always 200000000.

Example

If the basic system clock is a 24-hour clock that registers time in 1-second intervals, at 11:30 a.m., the reference

```
CALL SYSTEM_CLOCK (COUNT=C, COUNT_RATE=R, COUNT_MAX=M)
```

sets **C** = 11 x 3600 + 30 x 60 = 41400, **R** = 1, and **M** = 24 x 3600 − 1 = 86399.

Copyright © 1994 Thinking Machines Corporation

TINY

Returns the smallest positive number in the model representing numbers of the same type and kind type as the argument.

Syntax

```
TINY( X )
```

Kind

Inquiry function.

Argument

X A scalar or array of type **REAL**.

Result Type

Scalar of the same type and kind type as **X**.

Result Value

The result has the value $2^{e_{\min}-1}$, where $e_{\min}$ is as defined in Section 4.1.2 for the model representing numbers of the same type and kind type as **X**.

The result is one of the following values, depending on the kind type of the argument.

Data Type	Value
REAL*4	2^{-126}
REAL*8	2^{-1022}

Copyright © 1994 Thinking Machines Corporation

11 CMGL_global_to_physical Inquiry Procedure

This section contains the man page for the new global/local inquiry function discussed in Section 4.1.5.

Copyright © 1994 Thinking Machines Corporation

CMGL_global_to_physical

Given global array indices, provides the number of the corresponding physical node.

CM Fortran Syntax

```
CALL CMGL_GLOBAL_TO_PHYSICAL(LOCAL_ARRAY, G_INDEX, PN, L_INDEX)
```

C Syntax

```
PN = CMGL_global_to_physical(LOCAL_ARRAY, G_INDEX, L_INDEX);
```

Arguments

LOCAL_ARRAY A subarray (that is, the calling node's portion of a global parallel array). For C routines, the argument type is a CMRT array descriptor (**CMRT_desc_t**); for CM Fortran routines, it is the data type of **LOCAL_ARRAY**.

G_INDEX A one−dimensional serial integer array whose size is equal to the rank of **LOCAL_ARRAY**. It contains the coordinates of an element within the global array of which **LOCAL_ARRAY** is a subarray. In CM Fortran, **G_INDEX** is an **INTENT(IN)** argument.

PN In CM Fortran, the number of the physical node on which **G_INDEX** resides. **PN** is a scalar integer with a value in the range 0 through *number of nodes − 1*. In CM Fortran, **PN** is an **INTENT(OUT)** argument.

L_INDEX A one−dimensional serial integer array whose size is equal to the rank of **LOCAL_ARRAY**. It receives the coordinates within the subarray, **LOCAL_ARRAY**, equivalent to the element identified by **G_INDEX**. If the requested element is not in the subarray, the values in **L_INDEX** are undefined. In CM Fortran, **L_INDEX** is an **INTENT(OUT)** argument.

Version 2.2, October 1994
Copyright © 1994 Thinking Machines Corporation

Result

For CM Fortran, none. For C, an integer scalar, representing the number of the physical node.

Description

`CMGL_global_to_physical`, given a set of global coordinates within a local array, provides the number of the physical node containing the array element with those coordinates. All coordinates assume a 1-based array for CM Fortran code or a 0-based array for C code.

In CM Fortran, **`CMGL_global_to_physical`** assigns to **`PN`** the number of the physical node containing the coordinates specified in **`G_INDEX`**. In C, the function returns the number of the physical node. Both versions assign to **`L_INDEX`** the coordinates, within the subarray on that node, of the location specified in **`G_INDEX`**.

If **`G_INDEX`** is out of range, **`CMGL_global_to_physical`** returns −1 in **`PN`** and assigns −1 to **`L_INDEX`**.

`CMGL_global_to_physical` must be called only from local routines, and only on subarrays. It is a subroutine in CM Fortran, and a function returning an integer in C.

Copyright © 1994 Thinking Machines Corporation

Examples

Imagine a 16 x 16 CM Fortran global array named **A** divided across four nodes.

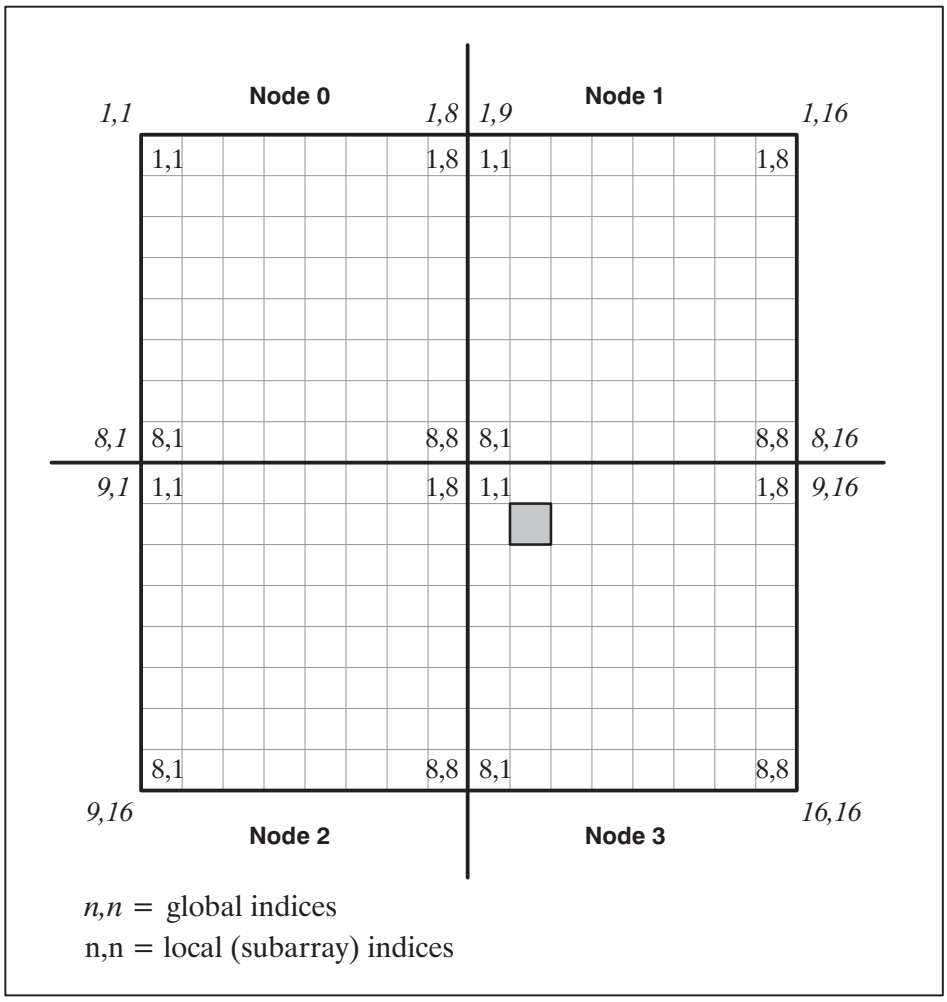

Figure 2. A 2D array and its four subarrays.

If **CMGL_global_to_physical** is called from code running on any of the four nodes and given the shaded square, 10,10, as **G_INDEX**, **PN** becomes 3 and **L_INDEX** becomes 2,2.

Version 2.2, October 1994
Copyright © 1994 Thinking Machines Corporation

The following example shows a way to use **CMGL_global_to_physical**:

```
C ********************************************************
C  Function name:         FIND_PREV_PN
C
C  Purpose:               Find out what PN is next to this
C                         one in the -X direction
C ********************************************************

       INTEGER FUNCTION FIND_PREV_PN(LOCAL_ARRAY)
       INTEGER LOCAL_ARRAY(:,:,:)
CMF$LAYOUT LOCAL_ARRAY(:NEWS,:NEWS,:NEWS)

       INTEGER L_INDEX(3), G_INDEX(3), PN
CMF$LAYOUT L_INDEX(:SERIAL), G_INDEX(:SERIAL)

C First get the global coordinate of our local origin:

       L_INDEX(0) = 1
       L_INDEX(1) = 1
       L_INDEX(2) = 1
       CALL CMGL_LOCAL_TO_GLOBAL(LOCAL_ARRAY, L_INDEX, G_INDEX)

C Adjust global coordinate:

       G_INDEX(0) = G_INDEX(0) - 1

C Now convert to 'local', though it'll be someone else's local,
C since G_INDEX(0) is not in this array.
C This returns PN==-1 if it's out of bounds.

       CALL CMGL_GLOBAL_TO_PHYSICAL(LOCAL_ARRAY, G_INDEX,
      &                             PN, L_INDEX)

       FIND_PREV_PN = PN
       RETURN
       END
```


Copyright © 1994 Thinking Machines Corporation

Error Conditions

Calling **`CMGL_global_to_physical`** on an array which is not a subarray (that is, it is not a global parallel array that was passed to the local routine as an argument), causes an error that the compiler cannot catch at compile time. As a result, such errors will probably cause your program to crash.

Calling the routine from a global program will cause a failure at link time.

Copyright © 1994 Thinking Machines Corporation

12 CMF_INIT

This section contains the man page for **CMF_INIT**, described in Section 4.1.6.

Copyright © 1994 Thinking Machines Corporation

CMF_INIT

Initializes the parallel processing unit in preparation for calling a CM Fortran subprogram from a serial program.

Fortran Syntax

```
CALL CMF_INIT(PADDING, AXISREORDER)
```

C Syntax

```
CMF_init(padding, axisreorder);
```

Arguments

PADDING A scalar logical (Fortran) or integer (C) that specifies whether the CM Fortran subprogram(s) to be linked in were compiled to expect vector-length padding in the allocation of CM arrays. The value **.TRUE.** in Fortran or nonzero in C signifies **-padding** (the **cmf** default); the value **.FALSE.** in Fortran or zero in C signifies **-nopadding**.

AXISREORDER A scalar logical (Fortran) or integer (C) that specifies whether the CM Fortran subprogram(s) to be linked in were compiled to expect axis reordering in the allocation of CM arrays. The value **.TRUE.** in Fortran or nonzero in C signifies **-axisreorder** (the **cmf** default); the value **.FALSE.** in Fortran or zero in C signifies **-noaxisreorder**.

Result

None.

Copyright © 1994 Thinking Machines Corporation

Description

CMF_INIT must be called to enable data parallel code to execute on the CM system. When the main program unit is compiled by **cmf** (by virtue of having the filename extension **.fcm** or **.FCM**), the compiler generates a call to this procedure. When the main program unit is compiled by a serial language compiler, the program must call **CMF_INIT** explicitly before calling any CM Fortran subroutine or function.

The arguments indicate certain switch settings used in the compilation of the CM Fortran units. These arguments are passed to the run-time system where they affect the layout of CM arrays. All CM Fortran subprograms must have been compiled with the switch settings indicated in the arguments. Any subsequent calls to **CMF_INIT** after the first one are ignored.

The **cmf** switches **-[no]padding** and **-[no]axisreorder** are used to influence performance, particularly when the program specifies detailed layouts for CM arrays. See the *CM-5 CM Fortran Performance Guide* for an explanation of the significance of these switches. The defaults are **-padding** and **-axisreorder**.

The requirement to call **CMF_INIT** and to identify the switch settings used in compiling the subprograms is the same regardless of whether the calling program is running globally (on the partition manager) or nodally (copies on each of the nodes).

For information on calling CM Fortran from C*, please see the *C* User's Guide*. C* programs need not call **CMF_init**, but C* Version 7.1 requires that the CM Fortran subprograms be compiled with **-padding** (the default).

Examples

These two examples appear on-line in **/usr/examples/cmfortran/f77_main_1** and **c_main_1**, along with **make** files for compilation and linking. See your site system administrator for the location if these files have been moved.

Copyright © 1994 Thinking Machines Corporation

A Fortran 77 program calling a CM Fortran subroutine:

```
      PROGRAM TST1                ! in tst1-f.f
      REAL A(1000)
      INTEGER I

      CALL CMF_INIT(.TRUE., .TRUE.)

      DO I = 1,1000
          A(I) = I
      END DO

      DO I = 1,901,100
          CALL MYCMFSUB(A(I))
      END DO

      PRINT 10, A
   10 FORMAT(10F6.2)
      END

      SUBROUTINE MYCMFSUB(A)      ! in tst1.fcm
      REAL A(100)
CMF$  LAYOUT A(:SERIAL)
      REAL A_CM(100), B
CMF$  LAYOUT A_CM(:NEWS)

      CALL CMF_FE_ARRAY_TO_CM(A_CM, A)
      B = SUM(A_CM)
      A_CM = (A_CM * A_CM) / B
      CALL CMF_FE_ARRAY_FROM_CM(A, A_CM)

      PRINT *, ' The sum is ', B
      END
```


Copyright © 1994 Thinking Machines Corporation

A C program calling a CM Fortran subroutine:

```
#include <stdio.h>                /* in tst1-c.c */

#define TRUE 1
#define FALSE 0

main()
{
      float a[1000];
      int i;

      CMF_init(TRUE,TRUE);

      for (i = 0; i < 1000; i++)
        a[i] = i+1;

      for (i = 0; i < 1000; i += 100)
        mycmfsub_(&a[i]);

      for (i = 0; i < 1000; i++)
        printf(" %f\n", a[i]);
}

      SUBROUTINE MYCMFSUB(A)      ! in tst1.fcm
      REAL A(100)
CMF$  LAYOUT A(:SERIAL)
      REAL A_CM(100), B
CMF$  LAYOUT A_CM(:NEWS)

      CALL CMF_FE_ARRAY_TO_CM(A_CM, A)
      B = SUM(A_CM)
      A_CM = (A_CM * A_CM) / B
      CALL CMF_FE_ARRAY_FROM_CM(A, A_CM)

      PRINT *, ' The sum is ', B
      END
```


Copyright © 1994 Thinking Machines Corporation