Naval Research Laboratory
Washington, DC 20375-5000

NRL Memorandum Report 6807

NRL Connection Machine Fortran Library

MICHAEL A. YOUNG

Signal Processing Branch
Acoustics Division

April 16, 1991

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	1991 April 16	Final

4. TITLE AND SUBTITLE

NRL Connection Machine Fortran Library

5. FUNDING NUMBERS

6. AUTHOR(S)

Michael A. Young

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Research Laboratory
Washington, DC 20375

8. PERFORMING ORGANIZATION REPORT NUMBER

NRL Memorandum
Report 6807

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The NRL Connection Machine Fortran Library consists of numerous mathematical routines coded in CM Fortran along with lower level routines written in Paris which manipulate data, plot data, and perform operations unavailable in the context of the CM Fortran language. Users are able to remain entirely within the CM Fortran programming environment while making calls to these library routines. Interfaces to the Framebuffer, a high resolution graphics device, and the DataVault, a high speed I/O channel, are available. A summary and demonstration of each library routine is provided.

14. SUBJECT TERMS

Connection Machine, Fortran

15. NUMBER OF PAGES
198

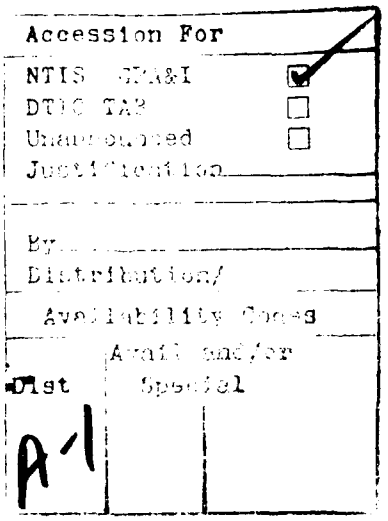
16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
unclassified	unclassified	unclassified	SAR

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18
298-102

CONTENTS



NRL CONNECTION MACHINE FORTRAN LIBRARY

1. INTRODUCTION

The Naval Research Lab (NRL) Fortran library on the Connection Machine (CM) [1] consists of numerous mathematical routines coded in CM Fortran [2], release 0.7, along with lower level routines written in Paris [3] which manipulate data, plot data, and perform operations unavailable in the context of the CM Fortran language. The contributing authors to this library package were Eric Hoffman, Michael Mascagni, Charles Del Vecchio, Robert Whaley, and Michael Young. CM Fortran consists of a mixture of serial and parallel array operations. Serial operations are executed by the front-end computer using its own memory and CPU. The parallel operations are executed on the CM-2 where each processor concurrently executes its own data point. Multidimensional arrays are allocated on the CM-2, one element per processor. Major array features that have been adapted from draft S8 of the ANSI Fortran 8x standard (x3.9−198x) [4] include array assignment, array constructors, and array sections. The *where* statement and *block where* construct are also featured. These allow the user to operate conditionally on array elements depending on their values.

The library routines fall into one of three categories: Paris Support Routines, Graphics Routines, or Linear Algebra Routines. The Paris Support Routines allow the user to perform operations on data that are currently not expressable in the context of the CM Fortran language. The Graphics Routines aid the user in displaying images on the framebuffer, a high speed graphics device. Lastly, the Linear Algebra Routines consist of frequently used mathematical operations. The purpose of each library routine is presented along with the parameters required, and an example call to the particular routine.

Users are able to remain entirely within the CM Fortran programming environment while making calls to these library routines. Interfaces to the Framebuffer, a high resolution graphics device [5], and the DataVault [6], a high speed I/O channel, are available. A user may access these routines from a CM Fortran program, by linking with the library, as shown in Figure 1.

```
cmf my_program.fcm -lnrlcmf
```

Fig. 1 — *Linking with the Library*

Some routines will be supplanted by the introduction of the CM Scientific Subroutine Library (CMSSL) [7]. The CMSSL software package is supplied directly by the manufacturer of the CM,

Manuscript approved January 15, 1991.

Thinking Machines Corporation(TMC).

2. Paris Support Routines

The Paris support routines use the paris/fortran interface [2] for many operations that are not expressable in CM Fortran. These *operations* include general communication between processors, and scanning functions, which combine calculation and communication.

2.1 DataVault

This package of routines is used to read and write information from the CM to the DataVault mass storage system. The DataVault provides a file system and permanent disk storage for the CM data. The routines provided are listed in Figure 2 with appropriate syntax. These routines work with arrays of type integer, logical, real and double precision. Arrays written to a file must conform, i.e. have the same shape [2].

```
dv_open(fd,path)
dv_close(fd)
dv_read(fd,buff)
dv_write(fd,buff)
dv_rewind(fd)
dv_lseek(fd,offset)

fd,offset : integer
path : character string
buff : integer, logical, or real array
```

Fig. 2 — *DataVault Routines Syntax*

Figure 3 demonstrates an example call to **dv_open**, which creates a file with an associated integer unit number "fd", and **dv_close**, which closes a file. Shown in Figure 4 are examples of reading and writing arrays to a DataVault file using **dv_read** and **dv_write**.

```
dv_open(fd,path)

integer fd

fd = 99
call dv_open(unit,'file_name')
```

```
dv_close(fd)

integer fd

call dv_close(fd)
```

Fig. 3 — *DataVault Open/Close Example*

Figure 5 illustrates how to manipulate the file pointer using subroutines **dv_rewind** and **dv_lseek**. Subroutine *dv_lseek* must be used with caution, since it moves the file pointer "off-

```
dv_read(fd,buff)

integer fd
integer buff1(128, 128)

fd = 99
call dv_read(fd, buff1)
```

```
dv_write(fd,buff)

integer fd
real buff2(32, 32, 32)

fd = 99
call dv_write(fd, buff2)
```

Fig. 4 — *DataVault Read/Write Example*

set" number of bits from its current position. This process allows the user to "lseek" to values of type logical within a file, since such values are stored as bits in CM Fortran.

```
dv_rewind(fd)

integer fd

call dv_rewind(fd)
```

```
dv_lseek(fd,offset)

integer fd,offset

offset = 10 * 32
call dv_lseek(fd,offset)
```

Fig. 5 — *DataVault Rewind/Lseek Example*

Table 1 lists the read/write rates for variable length blocks measured on an 8K CM-2 with a 10 Gigabyte DataVault at NRL [8].

Block Size	Read Rate	Write Rate
256K	1.3	0.6
512K	2.5	1.3
1024K	5.0	2.5
2048K	10.1	5.0
4096K	16.9	8.4
8192K	20.0	10.2

Table 1 —
8K CM-2
Performance in Mbytes/second

As shown in Table 1, it takes about the same time (200 ms) to read a 256K block as it takes to read a 2048K block. The same observation holds for writing the information to the DataVault. The largest possible block sizes will yield the most efficient use of the DataVault.

2.2 Gather/Scatter Routines

The Gather/Scatter routines are used to perform general communication between processors. They are particularly useful in data transfer between arrays of varying dimension. Figures 6 and 7 illustrate the appropriate syntax for the Gather and Scatter routines, respectively.

The gather package of routines provide a general gathering operation that is currently not expressable with the array constructs of CM Fortran. This gathering operation is needed when data must be exchanged between processors, requiring general interprocessor communication. Figure 8 compares the Fortran 77 code to the associated library call demonstrating the purpose of this routine.

The destination array may be n-dimensional and routines are provided for the source being 1, 2, 3, or 4 dimensional. The destination array(s) and index arrays must have the same subscript list. Routines are provided for gathering 1, 2, 3, or 4 arrays (which have the same index array(s)) at once to minimize the communication time involved. Figure 9 contains an example of the gather1_2d routine for a 1 dimensional destination *gathering* from a 2 dimensional source.

The scatter package of routines provide a general scattering operation that, similiar to the gather operation, is currently not expressable with the array constructs of CM Fortran. This scattering operation is needed when data must be sent to other processors, requiring general interprocessor communication. When multiple values are sent to the same processor, an add, max, or min combining operation is performed. Figure 10 compares the Fortran 77 code to the associated library call to illustrate the function of this routine.

The source array may be n-dimensional and routines are provided for the destination being 1, 2, or 3 dimensional. As with the gather routines, the source array and index array(s) must have the same subscript list. Figure 11 demonstrates the usage of the scatter_add_1 routine for a 2 dimensional source *scattering* to a 1 dimensional destination.

```
gather1_1d(dest,index_1,source_1d)
gather2_1d(dest1,dest2,index_1,source1_1d,source2_1d)
gather3_1d(dest1,dest2,dest3,index_1,source1_1d,source2_1d,
           source3_1d)
gather4_1d(dest1,dest2,dest3,dest4,index_1,source1_1d,source2_1d,
           source3_1d,source4_1d)

gather1_2d(dest,index_1,index_2,source_2d)
gather2_2d(dest1,dest2,index_1,index_2,source1_2d,source2_2d)
gather3_2d(dest1,dest2,dest3,index_1,index_2,source1_2d,
           source2_2d,source3_2d)
gather4_2d(dest1,dest2,dest3,dest4,index_1,index_2,source1_2d,
           source2_2d,source3_2d,source4_2d)

gather1_3d(dest,index_1,index_2,index_3,source_3d)
gather2_3d(dest1,dest2,index_1,index_2,index_3,source1_3d,
           source2_3d)
gather3_3d(dest1,dest2,dest3,index_1,index_2,index_3,source1_3d,
           source2_3d,source3_3d)
gather4_3d(dest1,dest2,dest3,dest4,index_1,index_2,index_3,
           source1_3d,source2_3d,source3_3d,source4_3d)

gather1_4d(dest,index_1,index_2,index_3,index_4,source_4d)
gather2_4d(dest1,dest2,index_1,index_2,index_3,index_4,source1_4d,
           source2_4d)
gather3_4d(dest1,dest2,dest3,index_1,index_2,index_3,index_4,
           source1_4d,source2_4d,source3_4d)
gather4_4d(dest1,dest2,dest3,dest4,index_1,index_2,index_3,index_4,
           source1_4d,source2_4d,source3_4d,source4_4d)

dest,dest1,dest2,dest3,dest4 : integer or real array (n-dimensional)
source_1d,source1_1d,source2_1d,source3_1d,source4_1d : integer/real array (1D)
source_2d,source1_2d,source2_2d,source3_2d,source4_2d : integer/real array (2D)
source_3d,source1_3d,source2_3d,source3_3d,source4_3d : integer/real array (3D)
source_4d,source1_4d,source2_4d,source3_4d,source4_4d : integer/real array (4D)
index_1,index_2,index_3,index_4 : integer array
```

Fig. 6 — *Gather Routines Syntax*

```
scatter_add_1(dest_1d,index_1,source)
scatter_add_2(dest_2d,index_1,index_2,source)
scatter_add_3(dest_3d,index_1,index_2,index_3,source)
scatter_min_1(dest_1d,index_1,source)
scatter_min_2(dest_2d,index_1,index_2,source)
scatter_min_3(dest_3d,index_1,index_2,index_3,source)
scatter_max_1(dest_1d,index_1,source)
scatter_max_2(dest_2d,index_1,index_2,source)
scatter_max_3(dest_3d,index_1,index_2,index_3,source)

source : integer/real array (n dimensional)
dest_1d : integer/real array (1 dimensional)
dest_2d : integer/real array (2 dimensional)
dest_3d : integer/real array (3 dimensional)
index_1,index_2,index_3 : integer array
```

Fig. 7 — *Scatter Routines Syntax*

Fortran 77

```
real a(m1,m2),c(n1,n2)
integer i,j,m1,m2,n1,n2
integer index_1(m1,m2),index_2(m1,m2)

do i=1,m1
do j=1,m2
a(i,j) = c(index_1(i,j),index_2(i,j))
enddo
enddo
```

Gather Routine

```
call gather1_2d(a,index_1,index_2,c)
```

Fig. 8 — *Gather Comparison*

This example gets the diagonal entries of the 2 dimensional matrix b and deposits them in the one dimensional vector a

```
integer m1,n1,n2
parameter(m1=4,n1=4,n2=4)

real b(n1,n2),a(m1)
integer index_1(m1),index_2(m1)

index_1 = [1:4]
index_2 = [1:4]
b(1,:) = [4.0, 6.0, 7.0, 9.0]
b(2,:) = [7.0, 3.0, 6.0, 5.0]
b(3,:) = [6.0, 5.0, 2.0, 9.0]
b(4,:) = [5.0, 7.0, 6.0, 1.0]

call gather1_2d(a,index_1,index_2,b)
```

input :

$$b = \begin{pmatrix} 4.0 & 6.0 & 7.0 & 9.0 \\ 7.0 & 3.0 & 6.0 & 5.0 \\ 6.0 & 5.0 & 2.0 & 9.0 \\ 5.0 & 7.0 & 6.0 & 1.0 \end{pmatrix}$$

output :

$$a = \begin{bmatrix} 4.0 \\ 3.0 \\ 2.0 \\ 1.0 \end{bmatrix}$$

Fig. 9 — *Gather Example for a 1D Destination Gathering from a 2D Source*

Fortran 77

```
real a(m1,m2),c(n1,n2)
integer i,j,m1,m2,n1,n2
integer index_1(n1,n2),index_2(n1,n2)

do i=1,n1
do j=1,n2
a(index_1(i,j),index_2(i,j)) =
a(index_1(i,j),index_2(i,j)) + c(i,j)
enddo
enddo
```

Scatter Routine

```
call scatter_add_2(a,index_1,index_2,c)
```

Fig. 10 — *Scatter Comparison*

In this example the row 1 values of c are sent and accumulated in a(1), values in rows 2 and 3 of c are accumulated in a(2), and values in row 4 of c are accumulated in a(3). No values are sent to a(4).

```
integer m1,n1,n2
parameter(m1=4,n1=4,n2=4)

real c(n1,n2),a(m1)
integer index_1(m1)

a = 0.0
index_1(1,:) = 1
index_1(2,:) = 2
index_1(3,:) = 2
index_1(4,:) = 3
c(1,:) = [4.0, 2.0, 6.0, 3.0]
c(2,:) = [1.0, 5.0, 7.0, 4.0]
c(3,:) = [9.0, 8.0, 6.0, 4.0]
c(4,:) = [4.0, 3.0, 7.0, 2.0]

call scatter_add_1(a,index_1,c)
```

input :

$$c = \begin{pmatrix} 4.0 & 2.0 & 6.0 & 3.0 \\ 1.0 & 5.0 & 7.0 & 4.0 \\ 9.0 & 8.0 & 6.0 & 4.0 \\ 4.0 & 3.0 & 7.0 & 2.0 \end{pmatrix}$$

output :

$$a = \begin{bmatrix} 15.0 \\ 44.0 \\ 16.0 \\ 0.0 \end{bmatrix}$$

Fig. 11 — *Scatter Example for a 2D Source Scattering to a 1D Destination*

2.3 Sprint Routines

The Sprint routines provide a simple interface to the indirect addressing hardware on the CM. This package of routines should be used for an array whose first or second dimension is serial. Each axis of an array may be set up to be parallel or serial on the CM with a layout compiler directive [2]. Parallel or serial referring to the programming context of that particular axis. To use this package the data in the array must be converted to a suitable format by calling the routine **begin_fast_array**. Upon finishing, the data must be returned to the normal CMF format by calling **end_fast_array**. Subroutine **fast_array_access** performs a retrieval operation and subroutine **fast_array_update** performs an updating or sending operation. The two dimensional versions of these routines are **fast_array_access_2d** and **fast_array_update_2d**. Figure 12 illustrates the appropriate syntax for the sprint routines.

```
begin_fast_array(array)
fast_array_access(dest,array,index)
fast_array_update(array,source,index)
fast_array_access_2d(dest,array_2,index1,index2)
fast_array_update_2d(array_2,source,index1,index2)
end_fast_array(array)

array : CM integer or real array (first dimension serial)
array_2 : CM integer or real array (first two dimensions serial)
dest,source : CM integer or real array
index,index1,index2 : CM integer array
```

Fig. 12 — *Sprint Syntax*

Figure 13 compares the CM Fortran code to the associated library calls required. The first forall corresponds to the *fast_array_access* routine and the second forall to the *fast_array_update* routine.

CM Fortran

```
real, array(30,128,128) :: array
real, array(128,128) :: dest,source
integer, array(128,128) :: index
integer i

forall (i=1:n) dest = array(index(i),:,:)
forall (i=1:n) array(index(i),:,:) = source
```

Sprint Routines

```
call fast_array_access(dest, array, index)
call fast_array_update(array, source, index)
```

Fig. 13 — *Sprint Comparison*

The parallel dimensions of the "source", "index", and "array" arguments must be conformable. Figure 14 provides an example setup and call of the two dimensional sprint routines.

```
This example demonstrates an access and update of an array using the sprint calls.

integer d1,d2,d3,d4
parameter (d1=13,d2=15,d3=128,d4=128)


integer a(d1,d2,d3,d4)
integer, array(d3,d4) :: i1,i2,b


cmf$ layout a(:serial,:serial,:news,:news)
cmf$ layout i1(:news,:news),i2(:news,:news)
cmf$ layout b(:news,:news)


c generate random numbers for i1 and i2
call CMF_random(i1,d1)
i1 = i1 + 1
call CMF_random(i2,d2)
i2 = i2 + 1


call begin_fast_array(a)
call fast_array_access_2d(b,a,i1,i2)
call fast_array_update_2d(a,b,i1,i2)
call end_fast_array(a)
```

Fig. 14 — *Sprint Example for two dimensions*

2.4 Table Lookup

The routines in this package are used to create a fast, integer or real, lookup table, extract values from the lookup table, and free up space when the lookup table is no longer needed. The routines provided are listed in Figure 15 with their appropriate syntax.

```
make_integer_lookup(fe_int_array,length)
make_real_lookup(fe_real_array,length)
make_lookup_cm(cm_source_array,cm_index,length,cm_mask)
lookup(cm_dest_array,lookup_table,cm_index,cm_mask)
free_lookup(lookup_table)

fe_int_array : front end integer array
fe_real_array : front end real array
cm_source_array : CM real or integer array
cm_dest_array : CM real or integer array
cm_index : CM integer array
cm_mask : CM logical array
length, lookup_table : integer
```

Fig. 15 — *Lookup Table Syntax*

The lookup table can be initialized from a front end array, residing in front end memory, or from a CM array, residing in CM memory. The function **make_real_lookup**, along with an associated integer version **make_integer_lookup**, create a lookup table with initial values taken from a front end array argument. The integer function **make_lookup_cm** creates a lookup table with initial values taken from a CM array argument. A CM array's data values are stored in CM memory whereas a front end array's data values are stored in front end memory.

Subroutine **lookup** extracts the value from the associated table entry. When the lookup table is no longer needed, subroutine **free_lookup** should be called to free up memory. This table is appropriate when the index for the lookup table is an array on the CM, and the lookup table is the same for every processor.

Figure 16 demonstrates the proper usage of these routines. Function *make_cm_lookup* creates a lookup table using initial values from the CM array "cm_source_array." The call to *make_lookup_cm* copies each element of "cm_source_array" to a location in the lookup table as specified by the "cm_index" corresponding to each source element. This occurs where the values of the logical mask "cm_mask" are true. This masking operation simply identifies which values of "cm_source_array" are to initialize the lookup table.

All selected elements must have a unique table index "cm_index" to place their table value "cm_source_array". Uninitialized elements of the lookup table will be set to 0. While the type of "cm_source_array" must be either real or integer, the user need only use this single function. Unlike the front end array initialization routines, *make_integer_lookup* and *make_real_lookup*, only one routine is needed when initializing the lookup table from a CM array (either of type real or integer); an integer is returned that identifies the table.

Values are extracted from the lookup table and assigned to the CM array "cm_dest_array" using subroutine *lookup*. This routine uses the table index "cm_index" to extract the corresponding table value from the lookup table "my_lookup_table." The extracted value is assigned to the CM array "cm_dest_array."

Finally, when the lookup has been accomplished, the memory used to store the lookup table must be deallocated by using routine *free_lookup*.

In this example, values from the cm_source_array initialize the lookup table and are then extracted and assigned to the cm_dest_array.

```
integer nproc
parameter(nproc=8)

integer my_lookup_table
integer, array(nproc) :: cm_index
real, array(nproc) :: cm_source_array, cm_dest_array
logical, array(nproc) :: cm_mask

cm_source_array = [2.0, 4.0, 8.0, 1.0, 7.0, 6.0, 3.0, 9.0]
cm_mask = .false.
cm_mask(1:nproc:2) = .true.
cm_index = [nproc:1:-1]

my_lookup_table = make_lookup_cm(cm_source_array,cm_index,nproc,cm_mask)
cm_dest_array = 20.0
call lookup(cm_dest_array,my_lookup_table,cm_index,cm_mask)
call free_lookup(my_lookup_table)
```

input :

$$cm\_source\_array = \begin{pmatrix} 2.0 & 4.0 & 8.0 & 1.0 & 7.0 & 6.0 & 3.0 & 9.0 \end{pmatrix}$$

$$cm\_index = \begin{pmatrix} 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 \end{pmatrix}$$

$$cm\_mask = \begin{pmatrix} T & F & T & F & T & F & T & F \end{pmatrix}$$

output :

$$cm\_dest\_array = \begin{bmatrix} 9.0 & 20.0 & 6.0 & 20.0 & 1.0 & 20.0 & 4.0 & 20.0 \end{bmatrix}$$

Fig. 16 — *Table Lookup Example*

2.5 Order

The **order** routine determines the ascending ordering of *real* or *integer* values in an array and generates an *integer* array of index values. The axis parameter is the array axis along which the ordering is required. Figure 17 contains the proper syntax for calling *order*. The source array of values, "cm_source_array", may be an integer or real array. The "cm_dest_array" is an integer array containing the indices of the source array in ascending order. The "cm_mask" is an integer array whose values specify whether the corresponding value of "cm_source_array" should be included for ordering. The elements of "cm_mask" should be set to 1 for inclusion or 0 for exclusion.

On return from *order*, the first element of "cm_dest_array" will contain the integer index of the source array's smallest value. Figure 18 contains a one dimensional example.

order(cm_dest_array,cm_source_array,axis,cm_mask)

cm_dest_array : integer array
cm_source_array : real or integer array
cm_mask : integer array
axis : integer

Fig. 17 — *Order Syntax*

In this example, values from the cm_source_array are ordered in ascending order and the index values are assinged to cm_dest_array. The values of cm_source_array are selected for ordering by setting the corresponding elements of cm_mask to 1.

```
integer nproc
parameter(nproc=8)

integer axis
real, array(nproc) :: cm_source_array
integer, array(nproc) :: cm_dest_array,cm_mask


cm_source_array = [4.0, 3.0, 6.0, 1.0, 2.0, 7.0, 9.0, 5.0]
cm_dest_array = 0
axis = 1
cm_mask = 1
call order(cm_dest_array,cm_source_array,axis,cm_mask)
```

input :

$$cm\_source\_array = \begin{pmatrix} 4.0 & 3.0 & 6.0 & 1.0 & 2.0 & 7.0 & 9.0 & 5.0 \end{pmatrix}$$

output :

$$cm\_dest\_array = \begin{bmatrix} 4 & 5 & 2 & 1 & 8 & 3 & 6 & 7 \end{bmatrix}$$

Fig. 18 — *Order Example - 1D*

2.6 Scan Functions

The functions contained in this package are used for parallel operations called "scans," which combine communication and calculation. These operations are very powerful in that they allow combining operations or calculations to be performed for each processor. A single dimension of a multidimensional array may be scanned. The combining operation may be numerically oriented (ADD, PRODUCT, MIN, or MAX) or logically oriented (OR, AND , or XOR). There is also a special scan, "copy scan," in which a value is simply copied to other processors, and a combining operation is not performed. A pleasant feature of the "scans" is that intermediate results are computed and stored. For example, if a total sum of all values of an array is needed, the intermediate values or partial sums would be computed, using a "sum scan." Figure 19 demonstrates what is meant by "partial sums."

In this example, values from vector a are added together and the partial sums are shown in vector b, the total sum of a is 38.

$$a = \begin{pmatrix} 2.0 & 5.0 & 3.0 & 5.0 & 7.0 & 8.0 & 2.0 & 6.0 \end{pmatrix}$$

$$b = \begin{bmatrix} 2.0 & 7.0 & 10.0 & 15.0 & 22.0 & 30.0 & 32.0 & 38.0 \end{bmatrix}$$

Fig. 19 — *Partial Sums Example*

Figure 20 contains a list of all scan functions and their associated syntax. The naming convention is such that the first part of the function name corresponds to the combining operation to be performed. It is also possible to start the scan anew at various points by assigning an element of the "sbit" array argument, representing the start bit, to true. When a "true" element of the array "sbit" is encountered, the scan is started over. Some of these operations are implemented in CM Fortran through reduction intrinsics. For example, the CM Fortran compiler generates a sum scan for the "sum" reduction intrinsic. However, the partial sums are not provided, only the total sum is available.

The **product_scan** is restricted to real values and the logical scans (**and_scan**, **or_scan**, **xor_scan**) accept only logical or integer arguments. **Sum_scan**, **max_scan**, and **min_scan** are restricted to integer or real values. **Copy_scan** is the only unrestricted scan because it does not perform a combining operation, simply a copy. The "dir" argument indicates the direction the scan is to be performed along. A value of "true" indicates an upward or forward direction whereas "false" indicates a downward or backward direction. The "dim" argument specifies which dimension the scan is to be performed along. In addition, a "mask" may be used to specify array elements which are not to be considered in the scan.

Figure 21 illustrates how the scanning process works for the *sum_scan*. Figure 22 illustrates an upward direction *product_scan*.

```
product_scan(real_result,real_source,dir,dim,sbit,mask)
sum_scan(result,source,dir,dim,sbit,mask)
max_scan(result,source,dir,dim,sbit,mask)
min_scan(result,source,dir,dim,sbit,mask)
or_scan(logint_result,logint_source,dir,dim,sbit,mask)
xor_scan(logint_result,logint_source,dir,dim,sbit,mask)
and_scan(logint_result,logint_source,dir,dim,sbit,mask)
copy_scan(any_result,any_source,dir,dim,sbit,mask)

real_result,real_source : real array
logint_result,logint_source : logical or integer array
any_result,any_source : integer, logical, or real array
result,source : integer or real array
dir : logical
dim : integer
sbit,mask : logical array
```

Fig. 20 — *Scan Syntax*

In this example, values from the source array are *summed* in the corresponding elements of the result array for a "dir" argument of upward and downward. Values of the result are set only for the true values of the mask. A new scan is started when both the sbit and mask are "true."

input :

$$source = \left(\begin{array}{cccccccccccccccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{array} \right)$$

$$mask = \left(\begin{array}{cccccccccccccccc} T & T & T & T & F & F & F & F & T & T & F & F & T & T & T & F \end{array} \right)$$

$$sbit = \left(\begin{array}{cccccccccccccccc} F & F & T & F & F & F & T & F & F & F & F & F & F & T & F & F \end{array} \right)$$

upward direction output :

$$result = \left[\begin{array}{cccccccccccccccc} 1 & 2 & 1 & 2 & - & - & - & - & 3 & 4 & - & - & 5 & 1 & 2 & - \end{array} \right]$$

downward direction output :

$$result = \left[\begin{array}{cccccccccccccccc} 3 & 2 & 1 & 5 & - & - & - & - & 4 & 3 & - & - & 2 & 1 & 1 & - \end{array} \right]$$

Fig. 21 — *General Sum Scan*

In this example, values from the source array are *multiplied* in the corresponding elements of the result array for a "dir" argument of true implying *upward* direction.

```
integer nproc
parameter (nproc = 8)

real, array(nproc) :: result,source
logical, array(nproc) ::sbit,cm_mask

sbit = .false.
cm_mask = .true.
cm_mask(4:6) = .false.
source = [.30, .40, 5.0, .50, 3.0, 4.0, .10, 2.0]
call product_scan(result,source,.true.,1,sbit,cm_mask)
```

input :

$$source = \begin{pmatrix} .30 & .40 & 5.0 & .50 & 3.0 & 4.0 & .10 & 2.0 \end{pmatrix}$$

$$cm\_mask = \begin{pmatrix} T & T & T & F & F & F & T & T \end{pmatrix}$$

$$sbit = \begin{pmatrix} F & F & F & F & F & F & F & F \end{pmatrix}$$

upward direction output :

$$result = \begin{bmatrix} .30 & .12 & .6 & - & - & - & .06 & .12 \end{bmatrix}$$

Fig. 22 — *Product Scan Example*

3. Graphics Routines

The graphics package of routines allows images to be displayed on the framebuffer through simple calls. This package allows a user to display images without dealing with the intricacies of the low level framebuffer calls themselves.

3.1 Framebuffer

The framebuffer routines are used to display pixels on the framebuffer. The process of displaying an image consists of initializing the framebuffer, setting an appropriate color map, displaying the pixels, and relinquishing the framebuffer.

The syntax of the routines in this package are listed in Figure 23.

```
init_fb(x_size,y_size)
release_frame_buffer()
set_color(color_id,red,green,blue)
plot_from_grid(color)
plot_x_y(x,y,color,mask)
plot_x_y_over(x,y,color,mask)

x_size,y_size,color_id,red,green,blue : integer
color : integer array
x,y : integer or real array
mask : logical array
```

Fig. 23 — *Framebuffer Routines Syntax*

Subroutines **init_fb** and **release_frame_buffer** initialize and release the framebuffer, respectively. Subroutine **set_color** allows the color map to be modified; the default color map is gray scale from 0 (black), to 255 (white). Red, green, and blue can each range from 0 to 255 giving a total of 16 million possible shades. Color 0 is the background color, and is usually left black (i.e. red=0, green=0, blue=0). Figure 24 demonstrates how to set a random color map for an image.

```
set_color(color_id, red, green, blue)

integer i

do i=1,32
call set_color(i, mod(irand(0),256),mod(irand(0),256),mod(irand(0),256))
enddo
```

Fig. 24 — *Framebuffer set_color*

Subroutine **plot_from_grid** updates a single pixel to a specified color for all selected processors,

which must be arranged in a 2 dimensional grid. Figure 25 illustrates a sample call.

```
plot_from_grid(color)

integer heat(512, 512)

heat = 128
call plot_from_grid(heat)
```

Fig. 25 — *Framebuffer plot_from_grid*

Subroutine **plot_x_y** sets each pixel to a specified color, clears the screen, and displays the color image. The **plot_x_y_over** routine is a variation of the *plot_x_y* routine in that it does not refresh the screen before displaying the image. Through the use of a logical mask, subsections of the actual image may be selected for display. Fig. 26 contains an example call to *plot_x_y*, with the resulting graphical output illustrated in Fig. 27.

```
In this example, the color map is filled with random values and a mask is used to select
a 256 by 256 grid in the upper left hand corner of the frambuffer.

integer npoints,xmax,ymax
parameter(npoints=65536,xmax=256,ymax=256)

integer i,irand
integer, array(npoints) :: x_position,y_position,color
logical, array(npoints) :: mask

call init_fb(xmax,ymax)
do i=1,32
call set_color(i, mod(irand(0),256), mod(irand(0),256), mod(irand(0),256))
enddo

** call cm random number generator
call cmf_random(x_position,xmax)
call cmf_random(y_position,ymax)
call cmf_random(color,32)

mask = x_position .le. 256 .and. y_position .le. 256

call plot_x_y_over(x_position,y_position,color,mask)
call release_frame_buffer()
```

Fig. 26 — *Framebuffer Code Example*

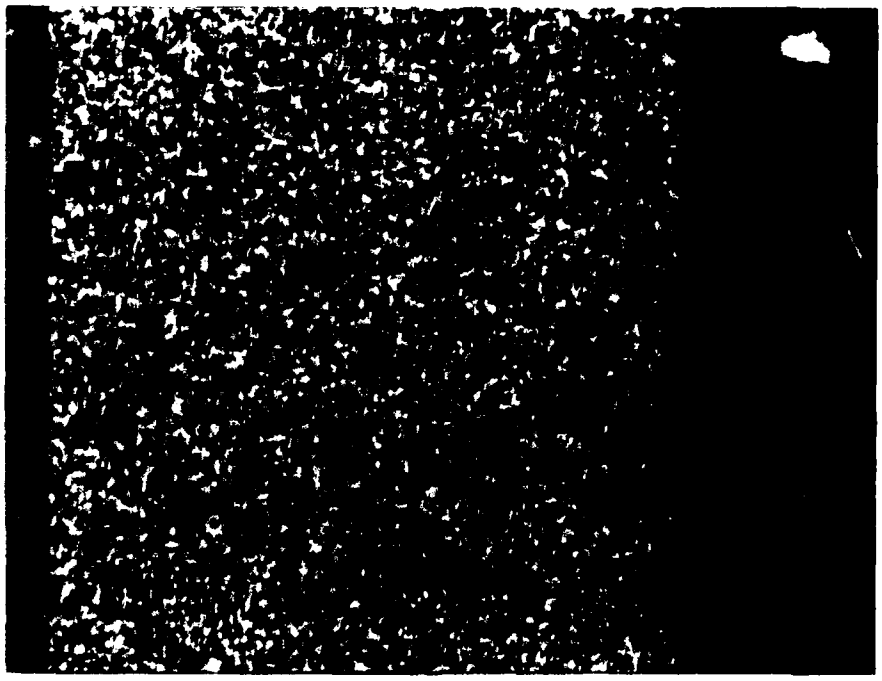


Fig. 27 — *Framebuffer Pictorial Example*

3.2 Plot

The plot package of routines provides an interface to the framebuffer similiar to unix plot functions. Coordinate values for all commands are reals and color values are integers. Mask values can be integer or logical, and contain a 1 in all array elements that participate in a draw operation. Figure 28 contains a list of all routines in this package along with the appropriate syntax. Plural subroutines take conformable array arguments.

```
openpl()
closepl()
erasepl()
set_color_value(color_id,red,green,blue)
set_text_size(size)
space(scalar_x1,scalar_y1,scalar_x2,scalar_y2)
line(scalar_x1,scalar_y1,scalar_x2,scalar_y2,color)
lines(array_x1,array_y1,array_x2,array_y2,color,mask)
circle(scalar_x,scalar_y,r,color)
circles(array_x,array_y,r,color,mask)
point(scalar_x,scalar_y,color)
points(array_x,array_y,color,mask)
label(string,length,scalar_x,scalar_y,color)

red,green,blue,size,length,color_id : integer
scalar_x,scalar_x1,scalar_x2 : real
scalar_y,scalar_y1,scalar_y2 : real
color : integer array
mask : logical array
string : character string
array_x,array_x1,array_x2 : real array
array_y,array_y1,array_y2 : real array
```

Fig. 28 — *Plot Routines Syntax*

Subroutines **openpl**, **closepl**, and **erasepl**, have an empty parameter list and simply attach, detach, and erase the framebuffer screen.

After attaching the framebuffer, the color map, text size, and window region must be set. The color values are set by using subroutine **set_color_value**, which sets an integer color value based on a red, green, and blue (rgb) triplet. Figure 29 sets the color number 14 to the rgb values 100, 200, 300.

Subroutine **set_text_size** allows the user to set the size of text used in labeling parts of the window. The six sizes currently available are 8, 10, 12, 14, 18, and 24 point corresponding to the input integer parameter values 0 through 5.

Subroutine **space** is used to define the window region for the framebuffer. The default setting is (0.0,0.0) in the upper left hand corner of the screen to (1023.0,1023.0) in the lower right. (x1,y1)

```
set_color_value(color,r,g,b)

set_color_value(14,100,200,300)
```

Fig. 29 — *Set_color_value Example*

defines the new upper left and (x2,y2) the new lower right. Figure 30 changes the window region to 2048 by 2048.

```
call space(0.0,0.0,2048.0,2048.0)
```

Fig. 30 — *Space Example*

Lines, circles, and points can be drawn on the framebuffer by using the subroutines line(s), circle(s), and point(s), respectively. Strings of text may be drawn by use of subroutine **label**. Figure 31 prints the word "hello" one line down and flush left of the screen with text size of 14 point.

```
label(string,length,x,y,color)

call set_color_value(1,255,125,125)
call set_text_size(3)
call label('hello',5,0,14,1)
```

Fig. 31 — *Label Example*

Figure 32 contains example code for drawing many concentric circles with the corresponding graphical output shown in Fig. 33.

```
In this example, many concentric circles are displayed

real x(512),y(512)
integer color(512)
logical mask(512)


color = 255
mask =.true.
x = [1:512]
y = 512.0
call openpl()
call circles(x,y,x,color,mask)
call closepl()
```

Fig. 32 — *Plot Code Example*

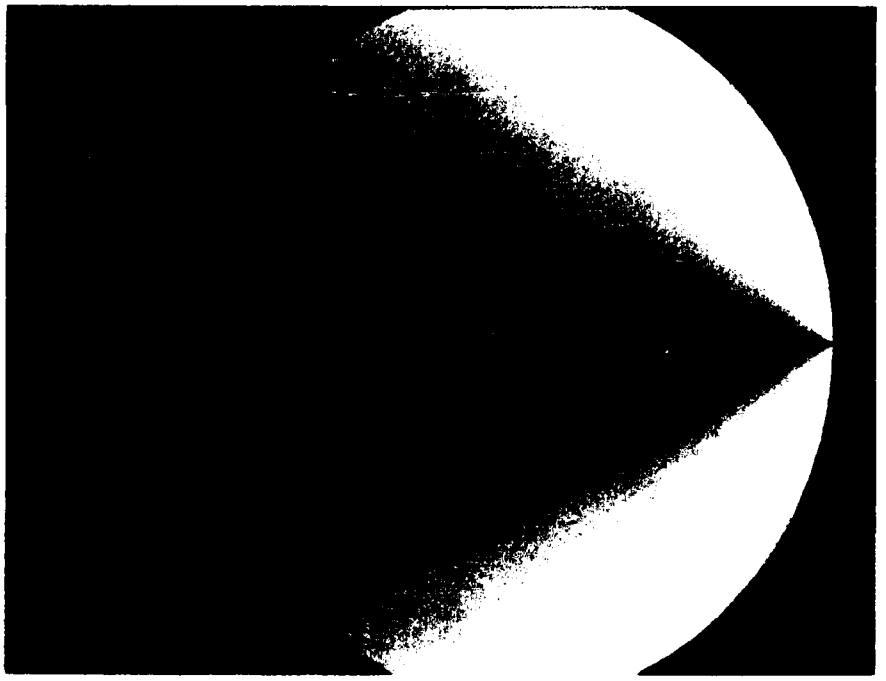


Fig. 33 — *Plot Pictorial Example*

3.3 Surface

The surface package of routines provides fast surface visualization for display on the framebuffer. The framebuffer must be initialized using subroutine *init_fb* as explained in Section 3.1 and Figure 23. Figure 34 demonstrates the proper syntax for calling the **surface**, **surface_over**, and **shade** routines.

```
surface (z,color,theta,phi)
surface_over (z,color,theta,phi)
shade (dest,z,theta,phi)

z : integer or real array (2 dimensional)
color,dest : integer array
theta,phi : real
```

Fig. 34 — *Surface Routines Syntax*

Subroutine *surface* displays a 3 dimensional surface using a 2 dimensional source and associated shading values. Routine *surface_over* is identical to the *surface* routine except that it plots the surface on top of the previous contents of the graphics buffer. The "z" value is a real two dimensional square array of elevations. "Color" is an array conformable to "z" which contains an integer color value from 0 to 255. As shown in Section 3.1 and 3.2, the appearance of these color values on the screen can be modified using the *set_color* or *set_color_value* routines. The "theta" and "phi" values are single real rotation values for the z-axis and x-axis respectively.

The *shade* routine is used to provide a shading value (without shadows) for a two dimensionsal array of elevations with a light source at the far right of the screen. It returns an integer color value from 0 to 255 in the destination array. As in subroutine surface, "theta" and "phi" are the rotations about the z and x axes.

Figure 35 illustrates an example for calling the surface library package of routines with the graphical output shown in Fig. 36.

```
integer len1,len2
parameter(len1=128,len2=128)

real, array(len1,len2) :: x,y,z
integer, array(len1,len2) :: color
integer i
real theta

x = spread([1:len1],2,len2)
y = spread([1:len2],1,len1)

z = cos(x*8.0*3.1415926365/128)+cos(y*8.0*3.1415926365/128)
z = z *30.0
theta = 0.0
call init_fb(256,256) ! initialize the framebuffer

do i=0,100
theta = theta + 0.1
call shade (color,z,theta,-0.7777) ! set up the shading values
call surface (z,color,theta,-0.7777) ! plot the surface
enddo
```

Fig. 35 — *Surface Code Example*

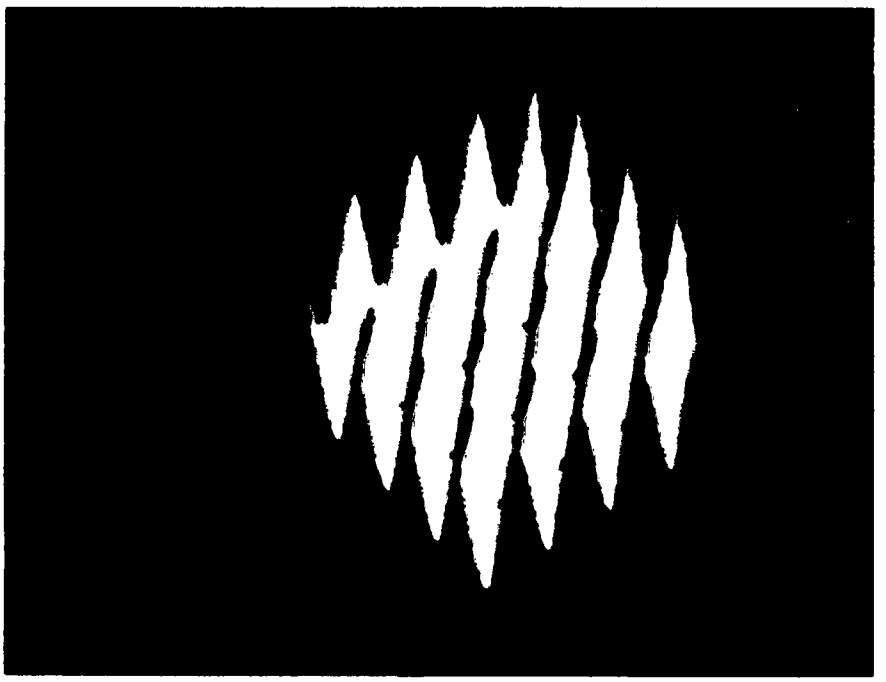


Fig. 36 — *Surface Pictorial Example*

4. Linear Algebra Routines

The linear algebra package of routines provides a user with specific examples of programming matrix operations in CM Fortran. The linear system routines will be supplanted by the CMSSL routines [7] upon release from TMC.

4.1 Polynomial Evaluation

The fast_poly package of routines are used to evaluate polynomials using Horner's rule. Each processor evaluates a polynomial based on a data point in that processor. The three subroutines in this package are used to set up the coefficients, evaluate the polynomial, and free up space when a coefficent is no longer needed. Figure 37 illustrates the proper syntax for these three routines, **make_horner_coef**, **eval_horner**, and **free_horner_coef**.

```
integer function make_horner_coeff(fe_coef_array,length)
eval_horner(cm_result,coef,cm_source)
free_horner_coef(coef)

cm_result,cm_source : real CM array
fe_coef_array : integer front end array
length,coef : integer
```

Fig. 37 — *Fast Poly Routines Syntax*

Figure 38 shows how to evaluate the polynomial $2.1x^3 + 0.5x^2 + 4x + 1.1$ in each CM processor.

In this example, values from the cm_source are evaluated using the fe_coef_array with results stored in cm_dest_array.

```
integer length,nproc
parameter(length=4,nproc=8)

real, array(nproc) :: cm_source,cm_result
integer, array(length) :: fe_coef_array
integer coef

fe_coef_array(1) = 2.1
fe_coef_array(2) = 0.5
fe_coef_array(3) = 4.0
fe_coef_array(4) = 1.1

cm_source = [1.0, 2.0, 0.0, 3.0, 0.0, 2.0, 1.0, 1.0]

coef = make_horner_coef(fe_coef_array) ! form coefficients
call eval_horner(cm_result,coef,cm_source) ! evaluate at each point
call free_horner(coef) ! free up space
```

input :

$$cm\_source = \begin{pmatrix} 1.0 & 2.0 & 0.0 & 3.0 & 0.0 & 2.0 & 1.0 & 1.0 \end{pmatrix}$$

output :

$$cm\_dest = \begin{bmatrix} 7.7 & 27.9 & 1.1 & 74.3 & 1.1 & 27.9 & 7.7 & 7.7 \end{bmatrix}$$

Fig. 38 — *Fast Polynomial Example*

4.2 Fast Fourier Transform

The purpose of the **fft** routine is to provide a CM Fortran interface for the Paris complex fast fourier transform routine. A complex array is constructed from the real and imaginary ("re_source" and "im_source") parts and passed into the Paris routine, whereupon the real and imaginary parts are extracted ("re_dest" and "im_dest") on return. The arrays may be laid out in *send* or *news* order in CM memory [3]. The send ordering is faster than the associated news ordering when an interface block is used [2]. Figure 39 demonstrates the proper syntax for calling the fft routine.

```
fft(re_dest,im_dest,re_source,im_source,operation)

re_dest,im_dest,re_source,im_source : real array
operation : integer array
```

Fig. 39 — *FFT Routine Syntax*

Figure 40 gives an example of calling the *fft* routine. The front end integer array "operation" indicates the transform (none = 0, forward = 1, or inverse = 2) to be performed along each axis. The size of the operation array is equal to the rank of the source/dest arrays. Arrays are laid out in *send* order through the use of the layout compiler directive.

```
integer n1,n2
parameter(n1=64,n2=64)


real, array(n1,n2) :: re_dest,im_dest,re_source,im_source
integer, array(2) :: operation

cmf$ layout re_dest(:send,:send),im_dest(:send,:send)
cmf$ layout re_source(:send,:send),im_source(:send,:send)
cmf$ layout operation(:serial)

interface

subroutine fft(re_dest,im_dest,re_source,im_source,operation)
integer n1,n2 parameter(n1=64,n2=64)
real, array(n1,n2) :: re_dest,im_dest,re_source,im_source
integer, array(2) :: operation

cmf$ layout re_dest(:send,:send),im_dest(:send,:send)
cmf$ layout re_source(:send,:send),im_source(:send,:send)
cmf$ layout operation(:serial)

end interface


** intialize input matrices


call cmf_random(re_source,0,0)
call cmf_random(im_source,0.0)

** perform forward transform along dimension 1
** and no transform along dimension 2

operation(1) = 1
operation(2) = 0
call fft(re_dest,im_dest,re_source,im_source,operation)
```

Fig. 40 — *Fast Fourier Transform Example*

4.3 Matrix Multiply

The purpose of the **matmul1** routine is to provide a CM Fortran interface for the Paris matrix multiply routine which allows the user access to a more efficient routine than the CM Fortran intrinsic function "matmul." However, the number of rows and columns must be a power of two and the number of elements of each matrix must be greater than or equal to the number of physical processors. If the above conditions are not satisfied, the CM Fortran intrinsic function matmul must be used. Figure 41 illustrates the proper syntax for calling *matmul1*. The parameters "matrix_a" and "matrix_b" are input and the result is returned in the CM array "result."

```
matmul1(matrix_a,matrix_b,result)

matrix_a,matrix_b,result : real array (2 dimensional)
```

Fig. 41 — *Matmul1 Routine Syntax*

Figure 42 demonstrates an example call to subroutine *matmul1*.

```
integer m,n,p
parameter(m=2,n=2,p=4)
real matrix_a(n,m),matrix_b(m,p),result(n,p)

result = 0.0
matrix_a(1,:) = 1.0
matrix_a(2,:) = 2.0

matrix_b(1,:) = [4.0, 8.0, 4.0, 7.0]
matrix_b(2,:) = [2.0, 1.0, 4.0, 3.0]

call matmul1(matrix_a,matrix_b,result)
input :
```

$$solve \begin{pmatrix} 1.0 & 1.0 \\ 2.0 & 2.0 \end{pmatrix} \begin{pmatrix} 4.0 & 8.0 & 4.0 & 7.0 \\ 2.0 & 1.0 & 4.0 & 3.0 \end{pmatrix}$$

output :

$$result = \begin{bmatrix} 6.0 & 9.0 & 8.0 & 10.0 \\ 12.0 & 18.0 & 16.0 & 20.0 \end{bmatrix}$$

Fig. 42 — *Matrix Multiplication Example*

4.4 Linear System Routines

Both of the routines in this library package will be included in the CMSSL library package. The linear system library package consists of a linear system solver and a matrix inversion routine. The library routine, **gauss**, solves a n x n system of linear equations using gaussian elimination. The input matrix, augmented by the forcing vector, is a n by n+1 system. On return the forcing vector is overwritten with the solution vector. The matrix is stored in the upper left hand corner of the 2 dimensional grid.

Subroutine **inv** computes the inverse of a square matrix, using a form of gaussian elimination. As with the gauss routine, the input matrix is stored in the upper left hand corner of the 2 dimensional grid. A work matrix, of size n by 2n, made up of the input matrix augmented by an identity matrix is used. The inverse solution overwrites the source matrix upon return from subroutine *inv*.

Figure 43 illustrates the proper calling procedure for subroutine *gauss* and *inv*. "N" is the dimension of the system and "matrix" is the actual linear system.

```
gauss(n,matrix)
inv(n,matrix)

matrix : real array (2 dimensional)
integer : n
```

Fig. 43 — *Linear System Routines Syntax*

Figure 44 and 45 contain examples for solving a linear system and computing the inverse of a matrix, respectively.

```
integer n
parameter(n=3)

real mat(n,n+1),forcing_vector(n),solution(n)

mat(1,1:n) = [4.0, 6.0, 2.0]
mat(2,1:n) = [1.0, 3.0, 5.0]
mat(3,1:n) = [7.0, 1.0, 8.0]
forcing_vector = [66.0, 66.0, 99.0]

mat(:,n+1) = forcing_vector
call gauss(n,mat)
solution = mat(:,n+1)
input :
```

$$solve \begin{pmatrix} 4.0 & 6.0 & 2.0 \\ 1.0 & 3.0 & 5.0 \\ 7.0 & 1.0 & 8.0 \end{pmatrix} \begin{pmatrix} s1 \\ s2 \\ s3 \end{pmatrix} = \begin{pmatrix} 66.0 \\ 66.0 \\ 99.0 \end{pmatrix}$$

output :

$$solution = \begin{bmatrix} 3.0 \\ 6.0 \\ 9.0 \end{bmatrix}$$

Fig. 44 — *Linear System Solver Example*

```
integer n
parameter(n=2)

real a(n,n)


a(1,:) = [7.0, 4.0]
a(2,:) = [6.0, 3.0]

call inv(n,a)
input :
```

$$\begin{pmatrix} 7.0 & 4.0 \\ 6.0 & 3.0 \end{pmatrix}$$

output :

$$solution = \begin{bmatrix} -1.0 & 4/3 \\ 2.0 & -7/3 \end{bmatrix}$$

Fig. 45 — *Matrix Inversion Example*

4.5 Tridiagonal Solver

Subroutine **tridiag** solves tridiagonal systems of equations, using a cyclic reduction algorithm [9], in log(n) time, n being the number of equations. The data is stored using four variables: the diagonal, the upper, the lower, and the right hand side.

If the data is configured as a one dimensional grid then a single system of equations is solved. If the data is configured as two or more dimensions, then M systems are solved simultaneously, where M is the product of the sizes of all dimensions greater than 1.

Figure 46 contains the syntax for the tridiagonal call.

```
tridiag(solution,lower,diagonal,upper,rhs)

solution,lower,diagonal,upper,rhs : real array
```

Fig. 46 — *Tridiagonal Solver Routine Syntax*

Figure 47 illustrates an example call to subroutine *tridiag*.

The tridiagonal routine works on diagonally dominant systems very well, but other tridiagonal systems may lead to inaccurate solutions due to the instability in the cyclic reduction algorithm

```
This example solves a tridiagonal system of order 4
integer size
parameter(size=4)

real, array(size) :: solution,lower,diagonal,upper,rhs

lower = 1.0
upper = 1.0
diagonal = 4.0
rhs(1) = 6.0
rhs(2) = 12.0
rhs(3) = 18.0
rhs(4) = 19.0

call tridiag(solution,lower,diagonal,upper,rhs)

input :
```

$$solve \begin{pmatrix} 4.0 & 1.0 & 0.0 & 0.0 \\ 1.0 & 4.0 & 1.0 & 0.0 \\ 0.0 & 1.0 & 4.0 & 1.0 \\ 0.0 & 0.0 & 1.0 & 4.0 \end{pmatrix} \begin{pmatrix} s1 \\ s2 \\ s3 \\ s4 \end{pmatrix} = \begin{pmatrix} 6.0 \\ 12.0 \\ 18.0 \\ 19.0 \end{pmatrix}$$

```
output :
```

$$solution = \begin{bmatrix} 1.0 \\ 2.0 \\ 3.0 \\ 4.0 \end{bmatrix}$$

Fig. 47 — *Tridiagonal Solver Example*

5. REFERENCES

1. "Connection Machine Model CM-2 Technical Summary," Thinking Machines Technical Report HA87-4, Apr. 1987.

2. "CM Fortran Reference Manual," Thinking Machines Corporation, Mar. 1990.

3. "CM Parallel Instruction Set," Thinking Machines Corporation, Oct. 1990.

4. "American National Standards for Information Systems Programming Language Fortran S8 (X3.9-198x) version 104," American National Standards Institute June 1987.

5. "Display Operations Reference Manual," Thinking Machines Corporation, 1989.

6. "CM DataVault Reference Manual," Thinking Machines Corporation, 1989.

7. "CM Scientific Subroutine Library," Thinking Machines Corporation, in press.

8. W. Anderson, G. Lamb, and W. Smith, "Acoustic Signal Processing on a Connection Machine," Naval Research Laboratory Technical Report, in press.

9. R. W. Hockney and C.R. Jesshope, *Parallel Computing* (Adam-Hilger, Bristol, 1981).

Appendix A
syntax

DataVault Routines

dv_open(fd,path)
dv_close(fd)
dv_read(fd,buff)
dv_write(fd,buff)
dv_rewind(fd)
dv_lseek(fd,offset)

fd,offset : integer
path : character string
buff : integer, logical, or real array

gather1_1d(dest,index_1,source_1d)
gather2_1d(dest1,dest2,index_1,source1_1d,source2_1d)
gather3_1d(dest1,dest2,dest3,index_1,source1_1d,source2_1d,
source3_1d)
gather4_1d(dest1,dest2,dest3,dest4,index_1,source1_1d,source2_1d,
source3_1d,source4_1d)
gather1_2d(dest,index_1,index_2,source_2d)

gather2_2d(dest1,dest2,index_1,index_2,source1_2d,source2_2d)
gather3_2d(dest1,dest2,dest3,index_1,index_2,source1_2d,
source2_2d,source3_2d)
gather4_2d(dest1,dest2,dest3,dest4,index_1,index_2,source1_2d,
source2_2d,source3_2d,source4_2d)
gather1_3d(dest,index_1,index_2,index_3,source_3d)

gather2_3d(dest1,dest2,index_1,index_2,index_3,source1_3d,
source2_3d)
gather3_3d(dest1,dest2,dest3,index_1,index_2,index_3,source1_3d,
source2_3d,source3_3d)
gather4_3d(dest1,dest2,dest3,dest4,index_1,index_2,index_3,
source1_3d,source2_3d,source3_3d,source4_3d)
gather1_4d(dest,index_1,index_2,index_3,index_4,source_4d)

gather2_4d(dest1,dest2,index_1,index_2,index_3,index_4,source1_4d,
source2_4d)
gather3_4d(dest1,dest2,dest3,index_1,index_2,index_3,index_4,
source1_4d,source2_4d,source3_4d)
gather4_4d(dest1,dest2,dest3,dest4,index_1,index_2,index_3,index_4,
source1_4d,source2_4d,source3_4d,source4_4d)

dest,dest1,dest2,dest3,dest4 : integer or real array (n-dimensional)
source_1d,source1_1d,source2_1d,source3_1d,source4_1d : integer/real array (1D)
source_2d,source1_2d,source2_2d,source3_2d,source4_2d : integer/real array (2D)
source_3d,source1_3d,source2_3d,source3_3d,source4_3d : integer/real array (3D)
source_4d,source1_4d,source2_4d,source3_4d,source4_4d : integer/real array (4D)
index_1,index_2,index_3,index_4 : integer array

Scatter Routines

scatter_add_1(dest_1d,index_1,source)
scatter_add_2(dest_2d,index_1,index_2,source)
scatter_add_3(dest_3d,index_1,index_2,index_3,source)
scatter_min_1(dest_1d,index_1,source)
scatter_min_2(dest_2d,index_1,index_2,source)
scatter_min_3(dest_3d,index_1,index_2,index_3,source)
scatter_max_1(dest_1d,index_1,source)
scatter_max_2(dest_2d,index_1,index_2,source)
scatter_max_3(dest_3d,index_1,index_2,index_3,source)

source : integer/real array (n dimensional)
dest_1d : integer/real array (1 dimensional)
dest_2d : integer/real array (2 dimensional)
dest_3d : integer/real array (3 dimensional)
index_1,index_2,index_3 : integer array

Sprint Routines

begin_fast_array(array)
fast_array_access(dest,array,index)
fast_array_update(array,source,index)
fast_array_access_2d(dest,array_2,index1,index2)
fast_array_update_2d(array_2,source,index1,index2)
end_fast_array(array)

array : CM integer or real array (first dimension serial)
array_2 : CM integer or real array (first two dimensions serial)
dest,source : CM integer or real array
index,index1,index2 : CM integer array

Table Lookup Routines

make_integer_lookup(fe_int_array,length)
make_real_lookup(fe_real_array,length)
make_lookup_cm(cm_source_array,cm_index,length,cm_mask)
lookup(cm_dest_array,lookup_table,cm_index,cm_mask)
free_lookup(lookup_table)

fe_int_array : front end integer array
fe_real_array : front end real array
cm_source_array : CM real or integer array
cm_dest_array : CM real or integer array
cm_index : CM integer array
cm_mask : CM logical array
length, lookup_table : integer

Order Routine

order(cm_dest_array,cm_source_array,axis,cm_mask)

cm_dest_array : integer array
cm_source_array : real or integer array
cm_mask : logical array
axis : integer

Scan Routines

product_scan(real_result,real_source,dir,dim,sbit,mask)
sum_scan(result,source,dir,dim,sbit,mask)
max_scan(result,source,dir,dim,sbit,mask)
min_scan(result,source,dir,dim,sbit,mask)
or_scan(logint_result,logint_source,dir,dim,sbit,mask)
xor_scan(logint_result,logint_source,dir,dim,sbit,mask)
and_scan(logint_result,logint_source,dir,dim,sbit,mask)
copy_scan(any_result,any_source,dir,dim,sbit,mask)

real_result,real_source : real array
logint_result,logint_source : logical or integer array
any_result,any_source : integer, logical, or real array
result,source : integer or real array
dir : logical
dim : integer
sbit,mask : logical array

Framebuffer Routines

init_fb(x_size,y_size)
release_frame_buffer()
set_color(color_id,red,green,blue)
plot_from_grid(color)
plot_x_y(x,y,color,mask)
plot_x_y_over(x,y,color,mask)

x_size,y_size,color_id,red,green,blue : integer
color : integer array
x,y : integer or real array
mask : logical array

Plot Routines

openpl()
closepl()
erasepl()
set_color_value(color_id,red,green,blue)
set_text_size(size)
space(scalar_x1,scalar_y1,scalar_x2,scalar_y2)
line(scalar_x1,scalar_y1,scalar_x2,scalar_y2,color)
lines(array_x1,array_y1,array_x2,array_y2,color,mask)
circle(scalar_x,scalar_y,r,color)
circles(array_x,array_y,r,color,mask)
point(scalar_x,scalar_y,color)
points(array_x,array_y,color,mask)
label(string,length,scalar_x,scalar_y,color)

red,green,blue,size,length,color_id : integer
scalar_x,scalar_x1,scalar_x2 : real
scalar_y,scalar_y1,scalar_y2 : real
color : integer array
mask : logical array
string : character string
array_x,array_x1,array_x2 : real array
array_y,array_y1,array_y2 : real array

Surface Routines

surface (z,color,theta,phi)
surface_over (z,color,theta,phi)
shade (dest,z,theta,phi)

z : integer or real array (2 dimensional)
color,dest : integer array
theta,phi : real

Polynomial Evaluation Routines

integer function make_horner_coeff(fe_coef_array,length)
eval_horner(cm_result,coef,cm_source)
free_horner_coef(coef)

cm_result,cm_source : real CM array
fe_coef_array : integer front end array
length,coef : integer

Fast Fourier Transform Routines

fft(re_dest,im_dest,re_source,im_source,operation)

re_dest,im_dest,re_source,im_source : real array
operation : integer array

Matrix Multiply Routines

matmul1(matrix_a,matrix_b,result)

matrix_a,matrix_b,result : real array (2 dimensional)

Linear Systems Routines

gauss(n,matrix)
inv(n,matrix)

matrix : real array (2 dimensional)
integer : n

Tridiagonal System Routines

tridiag(solution,lower,diagonal,upper,rhs)

solution,lower,diagonal,upper,rhs : real array

Appendix B
source

B1 DataVault Routines

```
      SUBROUTINE DV_OPEN(UNIT, PATH)
      CHARACTER*(*) PATH
      INTEGER UNIT
      CALL _DV_OPEN_C(PATH, UNIT)
      RETURN
      END

      SUBROUTINE DV_READ(UNIT, DEST)
      INTEGER UNIT, DEST
      INCLUDE '/usr/include/cm/paris-configuration-fort.h'
      INCLUDE '/usr/include/cm/CMF_defs.h'
      INTEGER DEST_TYP, DEST_VPS, LENGTH
      DEST_VPS = CMF_GET_VP_SET_ID(DEST)
      DEST_TYP = CMF_GET_DATA_TYPE(DEST)
      IF (DEST_TYP .EQ. CMF_FLOAT) THEN
      LENGTH = (CMF_GET_SIGNIFICAND_LEN(DEST) +
     +       CMF_GET_EXPONENT_LEN(DEST) + 1)
      CALL _DV_READ_C(UNIT,CMF_GET_FIELD_ID(DEST),LENGTH,DEST_VPS)
      ELSE IF ((DEST_TYP .EQ. CMF_U_INTEGER) .OR.
     +         (DEST_TYP .EQ. CMF_S_INTEGER))  THEN
      CALL _DV_READ_C(UNIT, CMF_GET_FIELD_ID(DEST), 32, DEST_VPS)
      ELSE IF (DEST_TYP .EQ. CMF_LOGICAL) THEN
      CALL _DV_READ_C(UNIT, CMF_GET_FIELD_ID(DEST), 1, DEST_VPS)
      ELSE IF (DEST_TYP .EQ. CMF_COMPLEX) THEN
      LENGTH = 2*(CMF_GET_SIGNIFICAND_LEN(DEST) +
     +       CMF_GET_EXPONENT_LEN(DEST) + 1)
      CALL _DV_READ_C(UNIT, CMF_GET_FIELD_ID(DEST), LENGTH, DEST_VPS)
      END IF
      CALL CMF_set_is_modified(dest,MODIF)
      RETURN
      END


      SUBROUTINE DV_WRITE(UNIT, SRC)
      INTEGER UNIT, SRC
      INCLUDE '/usr/include/cm/paris-configuration-fort.h'
      INCLUDE '/usr/include/cm/CMF_defs.h'
```

```
      INTEGER SRC_TYP, SRC_VPS, LENGTH
      SRC_VPS = CMF_GET_VP_SET_ID(SRC)
      SRC_TYP = CMF_GET_DATA_TYPE(SRC)
      IF (SRC_TYP .EQ. CMF_FLOAT) THEN
      LENGTH = (CMF_GET_SIGNIFICAND_LEN(SRC) +
     +       CMF_GET_EXPONENT_LEN(SRC) +1)
      CALL _DV_WRITE_C(UNIT,CMF_GET_FIELD_ID(SRC),LENGTH,SRC_VPS)
      ELSE IF ((SRC_TYP .EQ. CMF_U_INTEGER) .OR.
     +         (SRC_TYP .EQ. CMF_S_INTEGER))  THEN
      CALL _DV_WRITE_C(UNIT, CMF_GET_FIELD_ID(SRC), 32, SRC_VPS)
      ELSE IF (SRC_TYP .EQ. CMF_LOGICAL) THEN
      CALL _DV_WRITE_C(UNIT, CMF_GET_FIELD_ID(SRC), 1, SRC_VPS)
      ELSE IF (SRC_TYP .EQ. CMF_COMPLEX) THEN
      LENGTH = 2*(CMF_GET_SIGNIFICAND_LEN(SRC) +
     +       CMF_GET_EXPONENT_LEN(SRC) + 1)
      CALL _DV_WRITE_C(UNIT, CMF_GET_FIELD_ID(SRC), LENGTH, SRC_VPS)
      END IF
      RETURN
      END

      SUBROUTINE DV_CLOSE(UNIT)
      INTEGER UNIT
      CALL _DV_CLOSE_C(UNIT)
      RETURN
      END

      SUBROUTINE DV_REWIND(UNIT)
      INTEGER UNIT
      CALL _DV_LSEEK_C(UNIT,0)
      RETURN

      END

      SUBROUTINE DV_LSEEK(UNIT,OFFSET)
      INTEGER UNIT,OFFSET
      CALL _DV_LSEEK_INCR_C(UNIT,OFFSET)
      RETURN
      END


#include <string.h>
#include <stdio.h>
#include <cm/paris.h>
#include <cm/cmfs.h>
#include <cm/cm_file.h>
#include <cm/cm_errno.h>

#if defined(sparc)
```

```
# define DV_OPEN_C dv_open_c_
# define DV_LSEEK_C dv_lseek_c_
# define DV_LSEEK_INCR_C dv_lseek_incr_c_
# define DV_READ_C dv_read_c_
# define DV_WRITE_C dv_write_c_
# define DV_CLOSE_C dv_close_c_
#endif

#if defined(sparc)
struct ftn_string {char str[256]};
#else
struct ftn_string {short len; char * str};
#endif

char *for2c_string(forstr)
struct ftn_string *forstr;
{
  int i, true_len;
  char *name_dref, *temp, *rtn;
#if defined(sparc)
  i = 255;
#else
  i = forstr->len;
#endif
  name_dref = forstr->str;
  true_len = 0;
  temp = name_dref;
  while ((i--) && (*temp++ != ' ')) {
     true_len++; };
  rtn = temp = (char *) malloc(true_len+1);
  name_dref = forstr->str;
  i = true_len;
  *(temp+true_len)=0;
  while ((i--) && (*name_dref != ' ')) {
     *temp++ = *name_dref++; };
  /* add terminating null to end of string */
  *temp = 0;
  return(rtn);
}

void CMFS_perror();
int CMFS_errno;

void dv_open();
void dv_lseek();
void dv_read();
void dv_write();
void dv_close();
```

```
void dv_unlink();


#define S_ISUID 04000
#define S_ISGID 02000
#define S_ISVTX 01000
#define S_IRUSR 00400
#define S_IWUSR 00200
#define S_IXUSR 00100
#define S_IRWXG 00070
#define S_IRWXO 00007

static char *file_name[101];
static int units[101];

void DV_OPEN_C(name, open_return)
  int *open_return;
  struct ftn_string *name;
{
  file_name[*open_return] = for2c_string(name);
}

static void actually_open_the_file(unit)
int unit;
{
  int fd;
  fd = CMFS_open(file_name[unit], (CM_RDWR | CM_CREAT), (S_IRUSR | S_IWUSR),
    CM_physical_processors_limit, CM_user_number_of_processors_limit);
  free(file_name[unit]);
  file_name[unit] = 0;
  if (fd == -1) {
    fprintf(stderr, "OPEN FAILED.\n ERROR IS:");
    CMFS_perror("actually_open_the_file");
    exit(CMFS_errno);
  };
  units[unit] = fd;
}

void DV_LSEEK_C(fd,offset)
  int *fd,*offset;
{
  FILE *my_stderr;
  int lseek_return;
  if (file_name[*fd]) return;
  lseek_return = CMFS_lseek(units[*fd], *offset, 0);
  my_stderr = stderr;
  if (lseek_return == -1) {
    CMFS_perror("lseek");
```

```
    fprintf(my_stderr, "REWIND FAILED.\n ERROR IS:");
    exit(CMFS_errno);
  };
}


void DV_LSEEK_INCR_C(fd,offset)
  int *fd,*offset;
{
  FILE *my_stderr;
  int lseek_return;
  if (file_name[*fd]) return;
  lseek_return = CMFS_lseek(units[*fd], *offset, 1);
  my_stderr = stderr;
  if (lseek_return == -1) {
    CMFS_perror("lseek");
    fprintf(my_stderr, "REWIND FAILED.\n ERROR IS:");
    exit(CMFS_errno);
  };
}

void DV_READ_C(fd, buff, nbits, dest_vps)
  int *fd, *buff, *nbits, *dest_vps;
{
  FILE *my_stderr;
  int read_return;
  CM_set_vp_set(*dest_vps);
  if (file_name[*fd]) actually_open_the_file(*fd);
  read_return = CMFS_read_file_always(units[*fd], *buff, *nbits);
  my_stderr = stderr;
  if (read_return == -1) {
    fprintf(my_stderr, "READ FAILED.\n ERROR IS: ");
    CMFS_perror("read");
    exit(CMFS_errno);
  };
}

void DV_WRITE_C(fd, buff, nbits, dest_vps)
  int *fd, *buff,*nbits, *dest_vps;
{
  FILE *my_stderr;
  int write_return;
  CM_set_vp_set(*dest_vps);
  if (file_name[*fd]) actually_open_the_file(*fd);
  write_return = CMFS_write_file_always(units[*fd], *buff,*nbits);
  my_stderr = stderr;
  if (write_return == -1) {
    fprintf(my_stderr,"WRITE FAILED.\n ERROR IS:");
```

```
    CMFS_perror("write");
    exit(CMFS_errno);
  };
}

void DV_CLOSE_C(fd)
  int *fd;
 {
  FILE *my_stderr;
  int close_return;
  close_return = CMFS_close(units[*fd]);
  my_stderr = stderr;
  if (close_return == -1) {
    fprintf(my_stderr, "CLOSE FAILED.\n ERROR IS:");
    CMFS_perror("close");
    exit(CMFS_errno);
  };
}
```

B2 Gather/Scatter Routines

```
      subroutine gather1_1d(dest,index_1,source)

      include '/usr/include/cm/paris-configuration-fort.h'
      include '/usr/include/cm/CMF_defs.h'

c   compute  dest = source(index_1)
c   gets source (1-dimensional) for dest (n-dimensional)
c   the dest and index fields must be in the same vp set
c
c parameters
c    dest      : real destination field; n-dimensional
c    index_1   : integer field ; first index of source array
c    source    : real source field; 1-dimensional

      integer dest,dest_id,dest_vp_set,dest_geo
      integer index_1,index_1_id
      integer source,source_id,source_vp_set,source_geo
      integer get_field,length,temp,i1
      integer i,rank

      dest_id = cmf_get_field_id(dest)
      index_1_id = cmf_get_field_id(index_1)
      source_id = cmf_get_field_id(source)

      if (dest_id .eq. 0) then
         print *,
     +'Error, the dest argument to gather1_1d  is not on the CM'
         stop
      endif

      if (index_1_id .eq. 0) then
         print *,
     +'Error, the index_1 argument to gather1_1d is not on the CM'
         stop
      endif

      if (source_id .eq. 0) then
         print *,
     +'Error, the source argument to gather1_1d is not on the CM'
         stop
      endif

      dest_vp_set = cmf_get_vp_set_id(dest)
      dest_geo = cm_vp_set_geometry(dest_vp_set)

      source_vp_set = cmf_get_vp_set_id(source)
```

```
      source_geo = cm_vp_set_geometry(source_vp_set)

      call cm_set_vp_set(dest_vp_set)

      get_field = cm_allocate_stack_field(32)
      temp = cm_allocate_stack_field(32)
      i1 = cm_allocate_stack_field(32)

c      select context for destination

      call cm_set_context()
      call cm_my_news_coordinate_1l(temp,0,32)
      call cm_u_eq_constant_1l(temp,0,32)
      call cm_logand_context_with_test()

      rank = cm_geometry_rank(dest_geo)
      do i=1,rank-1
         length = cmf_get_axis_extent(dest,i-1)
         call CM_my_news_coordinate_1L(temp,i,32)
         call CM_u_lt_constant_1L(temp,length,32)
         call CM_logand_context_with_test()
      enddo

      call cm_u_move_zero_always_1l(get_field,32)
      call cm_u_subtract_constant_3_1l(i1,index_1_id,1,32)
      call cm_deposit_news_coordinate_1l(source_geo, get_field,1,
     +                                   i1,32)
      call cm_get_1l(dest_id,get_field,source_id,32)

      call cm_deallocate_stack_through(get_field)
      call CMF_set_is_modified(dest,MODIF)
      return
      end


      subroutine gather2_1d(dest1, dest2, index_1, source1, source2)

      include '/usr/include/cm/paris-configuration-fort.h'
      include '/usr/include/cm/CMF_defs.h'

c   compute  dest1 = source1(index_1)
c            dest2 = source2(index_1)
c
c   gets source (1-dimensional) for dest (n-dimensional)
c   the dest and index fields must be in the same vp set
c
c parameters
c    dest1      : real destination field; n-dimensional
c    dest2      : real destination field; n-dimensional
```

```
c     index_1     : integer field ; first index of source array
c     source1     : real source field; 1-dimensional
c     source2     : real source field; 1-dimensional

      integer dest1, dest2
      integer source1, source2
      integer dest1_id, dest2_id
      integer dest_vp_set,dest_geo
      integer index_1,index_1_id
      integer source1_id, source2_id
      integer source_vp_set,source_geo
      integer get_field,length,temp,i1
      integer i,rank

      integer source_temp_id, dest_temp_id

      dest1_id = cmf_get_field_id(dest1)
      dest2_id = cmf_get_field_id(dest2)

      index_1_id = cmf_get_field_id(index_1)
      source1_id = cmf_get_field_id(source1)
      source2_id = cmf_get_field_id(source2)

      if (dest1_id .eq. 0) then
         print *,
     +'Error, the dest1 argument to gather2_1d  is not on the CM'
         stop
      endif

      if (dest2_id .eq. 0) then
         print *,
     +'Error, the dest2 argument to gather2_1d  is not on the CM'
         stop
      endif

      if (index_1_id .eq. 0) then
         print *,
     +'Error, the index_1 argument to gather2_1d is not on the CM'
         stop
      endif

      if (source1_id .eq. 0) then
         print *,
     +'Error, the source1 argument to gather2_1d is not on the CM'
         stop
      endif

      if (source2_id .eq. 0) then
```

```
      print *,
     +'Error, the source2 argument to gather2_1d is not on the CM'
      stop
      endif

      dest_vp_set = cmf_get_vp_set_id(dest1)
      dest_geo = cm_vp_set_geometry(dest_vp_set)

      source_vp_set = cmf_get_vp_set_id(source1)
      source_geo = cm_vp_set_geometry(source_vp_set)
      call cm_set_vp_set(source_vp_set)
      source_temp_id = CM_allocate_stack_field(2*32)
      call CM_u_move_1L(source_temp_id, source1_id, 32)
      call CM_u_move_1L(source_temp_id + 32, source2_id, 32)

      call cm_set_vp_set(dest_vp_set)

      get_field = cm_allocate_stack_field(32)
      temp = cm_allocate_stack_field(32)
      i1 = cm_allocate_stack_field(32)
      call cm_set_context()
      dest_temp_id = CM_allocate_stack_field(2*32)

c      select context for destination

      call cm_my_news_coordinate_1l(temp,0,32)
      call cm_u_eq_constant_1l(temp,0,32)
      call cm_logand_context_with_test()

      rank = cm_geometry_rank(dest_   )
      do i=1,rank-1
         length = cmf_get_axis_extent(dest1,i-1)
         call CM_my_news_coordinate_1L(temp,i,32)
         call CM_u_lt_constant_1L(temp,length,32)
         call CM_logand_context_with_test()
      enddo

      call cm_u_move_zero_always_1l(get_field,32)
      call cm_u_subtract_constant_3_1l(i1,index_1_id,1,32)
      call cm_deposit_news_coordinate_1l(source_geo, get_field,1,
     +                                   i1,32)
      call cm_get_1l(dest_temp_id,get_field,source_temp_id,2*32)
      call CM_u_move_1L(dest1_id, dest_temp_id, 32)
      call CM_u_move_1L(dest2_id, dest_temp_id + 32, 32)

      call cm_deallocate_stack_through(dest_temp_id)
      call CMF_set_is_modified(dest1,MODIF)
      call CMF_set_is_modified(dest2,MODIF)
```

```
      return
      end



      subroutine gather3_1d(dest1, dest2, dest3, index_1,
     +                      source1, source2, source3)

      include '/usr/include/cm/paris-configuration-fort.h'
      include '/usr/include/cm/CMF_defs.h'

c   compute  dest = source(index_1)
c   gets source (1-dimensional) for dest (n-dimensional)
c   the dest and index fields must be in the same vp set
c
c parameters
c    dest1      : real destination field; n-dimensional
c    dest2      : real destination field; n-dimensional
c    dest3      : real destination field; n-dimensional
c    index_1    : integer field ; first index of source array
c    source1    : real source field; 1-dimensional
c    source2    : real source field; 1-dimensional
c    source3    : real source field; 1-dimensional

      integer dest1, dest2, dest3
      integer source1, source2, source3
      integer dest1_id, dest2_id, dest3_id
      integer dest_vp_set,dest_geo
      integer index_1,index_1_id
      integer source1_id, source2_id, source3_id
      integer source_vp_set,source_geo
      integer get_field,length,temp,i1
      integer i,rank

      integer source_temp_id, dest_temp_id

      dest1_id = cmf_get_field_id(dest1)
      dest2_id = cmf_get_field_id(dest2)
      dest3_id = cmf_get_field_id(dest3)

      index_1_id = cmf_get_field_id(index_1)
      source1_id = cmf_get_field_id(source1)
      source2_id = cmf_get_field_id(source2)
      source3_id = cmf_get_field_id(source3)

      if (dest1_id .eq. 0) then
         print *,
     +'Error, the dest1 argument to gather3_1d  is not on the CM'
         stop
```

```
 endif

 if (dest2_id .eq. 0) then
    print *,
+'Error, the dest2 argument to gather3_1d  is not on the CM'
    stop
 endif

 if (dest3_id .eq. 0) then
    print *,
+'Error, the dest3 argument to gather3_1d  is not on the CM'
    stop
 endif

 if (index_1_id .eq. 0) then
    print *,
+'Error, the index_1 argument to gather3_1d is not on the CM'
    stop
 endif

 if (source1_id .eq. 0) then
    print *,
+'Error, the source1 argument to gather3_1d is not on the CM'
    stop
 endif

 if (source2_id .eq. 0) then
    print *,
+'Error, the source2 argument to gather3_1d is not on the CM'
    stop
 endif

 if (source3_id .eq. 0) then
    print *,
+'Error, the source3 argument to gather3_1d is not on the CM'
    stop
 endif

 dest_vp_set = cmf_get_vp_set_id(dest1)
 dest_geo = cm_vp_set_geometry(dest_vp_set)

 source_vp_set = cmf_get_vp_set_id(source1)
 source_geo = cm_vp_set_geometry(source_vp_set)
 call cm_set_vp_set(source_vp_set)
 source_temp_id = CM_allocate_stack_field(3*32)
 call CM_u_move_1L(source_temp_id, source1_id, 32)
 call CM_u_move_1L(source_temp_id + 32, source2_id, 32)
 call CM_u_move_1L(source_temp_id + 64, source3_id, 32)
```

```
      call cm_set_vp_set(dest_vp_set)

      get_field = cm_allocate_stack_field(32)
      temp = cm_allocate_stack_field(32)
      i1 = cm_allocate_stack_field(32)
      call cm_set_context()
      dest_temp_id = CM_allocate_stack_field(3*32)

c      select context for destination

      call cm_my_news_coordinate_1l(temp,0,32)
      call cm_u_eq_constant_1l(temp,0,32)
      call cm_logand_context_with_test()

      rank = cm_geometry_rank(dest_geo)
      do i=1,rank-1
         length = cmf_get_axis_extent(dest1,i-1)
         call CM_my_news_coordinate_1L(temp,i,32)
         call CM_u_lt_constant_1L(temp,length,32)
         call CM_logand_context_with_test()
      enddo

      call cm_u_move_zero_always_1l(get_field,32)
      call cm_u_subtract_constant_3_1l(i1,index_1_id,1,32)
      call cm_deposit_news_coordinate_1l(source_geo, get_field,1,
     +                                   i1,32)
      call cm_get_1l(dest_temp_id,get_field,source_temp_id,3*32)
      call CM_u_move_1L(dest1_id, dest_temp_id, 32)
      call CM_u_move_1L(dest2_id, dest_temp_id + 32, 32)
      call CM_u_move_1L(dest3_id, dest_temp_id + 64, 32)

      call cm_deallocate_stack_through(dest_temp_id)
      call CMF_set_is_modified(dest1,MODIF)
      call CMF_set_is_modified(dest2,MODIF)
      call CMF_set_is_modified(dest3,MODIF)
      return
      end



      subroutine gather4_1d(dest1, dest2, dest3, dest4, index_1,
     +                      source1, source2, source3, source4)

      include '/usr/include/cm/paris-configuration-fort.h'
      include '/usr/include/cm/CMF_defs.h'

c   compute  dest = source(index_1)
c   gets source (1-dimensional) for dest (n-dimensional)
```

```
c   the dest and index fields must be in the same vp set
c
c parameters
c    dest1      : real destination field; n-dimensional
c    dest2      : real destination field; n-dimensional
c    dest3      : real destination field; n-dimensional
c    dest4      : real destination field; n-dimensional
c    index_1    : integer field ; first index of source array
c    source1    : real source field; 1-dimensional
c    source2    : real source field; 1-dimensional
c    source3    : real source field; 1-dimensional
c    source4    : real source field; 1-dimensional

      integer dest1, dest2, dest3, dest4
      integer source1, source2, source3, source4
      integer dest1_id, dest2_id, dest3_id, dest4_id
      integer dest_vp_set,dest_geo
      integer index_1,index_1_id
      integer source1_id, source2_id, source3_id, source4_id
      integer source_vp_set,source_geo
      integer get_field,length,temp,i1
      integer i,rank

      integer source_temp_id, dest_temp_id

      dest1_id = cmf_get_field_id(dest1)
      dest2_id = cmf_get_field_id(dest2)
      dest3_id = cmf_get_field_id(dest3)
      dest4_id = cmf_get_field_id(dest4)

      index_1_id = cmf_get_field_id(index_1)
      source1_id = cmf_get_field_id(source1)
      source2_id = cmf_get_field_id(source2)
      source3_id = cmf_get_field_id(source3)
      source4_id = cmf_get_field_id(source4)

      if (dest1_id .eq. 0) then
         print *,
     +'Error, the dest1 argument to gather4_1d  is not on the CM'
         stop
      endif

      if (dest2_id .eq. 0) then
         print *,
     +'Error, the dest2 argument to gather4_1d  is not on the CM'
         stop
      endif
```

```
 if (dest3_id .eq. 0) then
    print *,
+'Error, the dest3 argument to gather4_1d  is not on the CM'
    stop
 endif

 if (dest4_id .eq. 0) then
    print *,
+'Error, the dest4 argument to gather4_1d  is not on the CM'
    stop
 endif


 if (index_1_id .eq. 0) then
    print *,
+'Error, the index_1 argument to gather4_1d is not on the CM'
    stop
 endif

 if (source1_id .eq. 0) then
    print *,
+'Error, the source1 argument to gather4_1d is not on the CM'
    stop
 endif

 if (source2_id .eq. 0) then
    print *,
+'Error, the source2 argument to gather4_1d is not on the CM'
    stop
 endif

 if (source3_id .eq. 0) then
    print *,
+'Error, the source3 argument to gather4_1d is not on the CM'
    stop
 endif

 if (source4_id .eq. 0) then
    print *,
+'Error, the source4 argument to gather4_1d is not on the CM'
    stop
 endif

 dest_vp_set = cmf_get_vp_set_id(dest1)
 dest_geo = cm_vp_set_geometry(dest_vp_set)

 source_vp_set = cmf_get_vp_set_id(source1)
 source_geo = cm_vp_set_geometry(source_vp_set)
```

```
      call cm_set_vp_set(source_vp_set)
      source_temp_id = CM_allocate_stack_field(4*32)
      call CM_u_move_1L(source_temp_id, source1_id, 32)
      call CM_u_move_1L(source_temp_id + 32, source2_id, 32)
      call CM_u_move_1L(source_temp_id + 64, source3_id, 32)
      call CM_u_move_1L(source_temp_id + 96, source4_id, 32)

      call cm_set_vp_set(dest_vp_set)

      get_field = cm_allocate_stack_field(32)
      temp = cm_allocate_stack_field(32)
      i1 = cm_allocate_stack_field(32)
      call cm_set_context()
      dest_temp_id = CM_allocate_stack_field(4*32)

c      select context for destination

      call cm_my_news_coordinate_1l(temp,0,32)
      call cm_u_eq_constant_1l(temp,0,32)
      call cm_logand_context_with_test()

      rank = cm_geometry_rank(dest_geo)
      do i=1,rank-1
         length = cmf_get_axis_extent(dest1,i-1)
         call CM_my_news_coordinate_1L(temp,i,32)
         call CM_u_lt_constant_1L(temp,length,32)
         call CM_logand_context_with_test()
      enddo

      call cm_u_move_zero_always_1l(get_field,32)
      call cm_u_subtract_constant_3_1l(i1,index_1_id,1,32)
      call cm_deposit_news_coordinate_1l(source_geo, get_field,1,
     +                                   i1,32)
      call cm_get_1l(dest_temp_id,get_field,source_temp_id,4*32)
      call CM_u_move_1L(dest1_id, dest_temp_id, 32)
      call CM_u_move_1L(dest2_id, dest_temp_id + 32, 32)
      call CM_u_move_1L(dest3_id, dest_temp_id + 64, 32)
      call CM_u_move_1L(dest4_id, dest_temp_id + 96, 32)

      call cm_deallocate_stack_through(dest_temp_id)
      call CMF_set_is_modified(dest1,MODIF)
      call CMF_set_is_modified(dest2,MODIF)
      call CMF_set_is_modified(dest3,MODIF)
      call CMF_set_is_modified(dest4,MODIF)
      return
      end
```

```
      subroutine gather1_2d(dest,index_1,index_2,source)

      include '/usr/include/cm/paris-configuration-fort.h'
      include '/usr/include/cm/CMF_defs.h'

c   compute  dest = source(index_1,index_2)
c   gets source (2-dimensional) for dest (n-dimensional)
c   the dest and index fields must be in the same vp set
c
c parameters
c    dest       : real destination field; n-dimensional
c    index_1    : integer field ; first index of source array
c    index_2    : integer field ; second index of source array
c    source     : real source field; 2-dimensional

      integer dest,dest_id,dest_vp_set,dest_geo
      integer index_1,index_1_id
      integer index_2,index_2_id
      integer source,source_id,source_vp_set,source_geo
      integer get_field,length,temp,i1,i2
      integer i,rank

      dest_id = cmf_get_field_id(dest)
      index_1_id = cmf_get_field_id(index_1)
      index_2_id = cmf_get_field_id(index_2)
      source_id = cmf_get_field_id(source)

      if (dest_id .eq. 0) then
         print *,
     +'Error, the dest argument to gather1_2d  is not on the CM'
         stop
      endif

      if (index_1_id .eq. 0) then
         print *,
     +'Error, the index_1 argument to gather1_2d is not on the CM'
         stop
      endif

      if (index_2_id .eq. 0) then
         print *,
     +'Error, the index_2 argument to gather1_2d is not on the CM'
         stop
      endif

      if (source_id .eq. 0) then
         print *,
     +'Error, the source argument to gather1_2d is not on the CM'
```

```
      stop
    endif

    dest_vp_set = cmf_get_vp_set_id(dest)
    dest_geo = cm_vp_set_geometry(dest_vp_set)

    source_vp_set = cmf_get_vp_set_id(source)
    source_geo = cm_vp_set_geometry(source_vp_set)

    call cm_set_vp_set(dest_vp_set)

    get_field = cm_allocate_stack_field(32)
    temp = cm_allocate_stack_field(32)
    i1 = cm_allocate_stack_field(32)
    i2 = cm_allocate_stack_field(32)

c     select context for destination

    call cm_set_context()
    call cm_my_news_coordinate_1l(temp,0,32)
    call cm_u_eq_constant_1l(temp,0,32)
    call cm_logand_context_with_test()

    rank = cm_geometry_rank(dest_geo)
    do i=1,rank-1
       length = cmf_get_axis_extent(dest,i-1)
       call CM_my_news_coordinate_1L(temp,i,32)
       call CM_u_lt_constant_1L(temp,length,32)
       call CM_logand_context_with_test()
    enddo

    call cm_u_move_zero_always_1l(get_field,32)
    call cm_u_subtract_constant_3_1l(i1,index_1_id,1,32)
    call cm_u_subtract_constant_3_1l(i2,index_2_id,1,32)
    call cm_deposit_news_coordinate_1l(source_geo, get_field,1,
   +                                   i1,32)
    call cm_deposit_news_coordinate_1l(source_geo, get_field,2,
   +                                   i2,32)
    call cm_get_1l(dest_id,get_field,source_id,32)

    call cm_deallocate_stack_through(get_field)
    call CMF_set_is_modified(dest,MODIF)
    return
    end



    subroutine gather2_2d(dest1, dest2, index_1, index_2,
   +                      source1, source2)
```

```
      include '/usr/include/cm/paris-configuration-fort.h'
      include '/usr/include/cm/CMF_defs.h'

c   compute  dest1 = source1(index_1,index_2)
c            dest2 = source2(index_1,index_2)
c
c   gets source (2-dimensional) for dest (n-dimensional)
c   the dest and index fields must be in the same vp set
c
c parameters
c    dest1      : real destination field; n-dimensional
c    dest2      : real destination field; n-dimensional
c    index_1    : integer field ; first index of source array
c    index_2    : integer field ; second index of source array
c    source1    : real source field; 2-dimensional
c    source2    : real source field; 2-dimensional

      integer dest1, dest2
      integer source1, source2
      integer dest1_id, dest2_id
      integer dest_vp_set,dest_geo
      integer index_1,index_1_id
      integer index_2,index_2_id
      integer source1_id, source2_id
      integer source_vp_set,source_geo
      integer get_field,length,temp,i1,i2
      integer i,rank

      integer source_temp_id, dest_temp_id

      dest1_id = cmf_get_field_id(dest1)
      dest2_id = cmf_get_field_id(dest2)

      index_1_id = cmf_get_field_id(index_1)
      index_2_id = cmf_get_field_id(index_2)
      source1_id = cmf_get_field_id(source1)
      source2_id = cmf_get_field_id(source2)

      if (dest1_id .eq. 0) then
         print *,
     +'Error, the dest1 argument to gather2_2d  is not on the CM'
         stop
      endif

      if (dest2_id .eq. 0) then
         print *,
     +'Error, the dest2 argument to gather2_2d  is not on the CM'
```

```
      stop
    endif

    if (index_1_id .eq. 0) then
      print *,
     +'Error, the index_1 argument to gather2_2d is not on the CM'
      stop
    endif

    if (index_2_id .eq. 0) then
      print *,
     +'Error, the index_2 argument to gather2_2d is not on the CM'
      stop
    endif

    if (source1_id .eq. 0) then
      print *,
     +'Error, the source1 argument to gather2_2d is not on the CM'
      stop
    endif

    if (source2_id .eq. 0) then
      print *,
     +'Error, the source2 argument to gather2_2d is not on the CM'
      stop
    endif

    dest_vp_set = cmf_get_vp_set_id(dest1)
    dest_geo = cm_vp_set_geometry(dest_vp_set)

    source_vp_set = cmf_get_vp_set_id(source1)
    source_geo = cm_vp_set_geometry(source_vp_set)
    call cm_set_vp_set(source_vp_set)
    source_temp_id = CM_allocate_stack_field(2*32)
    call CM_u_move_1L(source_temp_id, source1_id, 32)
    call CM_u_move_1L(source_temp_id + 32, source2_id, 32)

    call cm_set_vp_set(dest_vp_set)

    get_field = cm_allocate_stack_field(32)
    temp = cm_allocate_stack_field(32)
    i1 = cm_allocate_stack_field(32)
    i2 = cm_allocate_stack_field(32)
    call cm_set_context()
    dest_temp_id = CM_allocate_stack_field(2*32)

c     select context for destination
```

```
      call cm_my_news_coordinate_1l(temp,0,32)
      call cm_u_eq_constant_1l(temp,0,32)
      call cm_logand_context_with_test()

      rank = cm_geometry_rank(dest_geo)
      do i=1,rank-1
         length = cmf_get_axis_extent(dest1,i-1)
         call CM_my_news_coordinate_1L(temp,i,32)
         call CM_u_lt_constant_1L(temp,length,32)
         call CM_logand_context_with_test()
      enddo

      call cm_u_move_zero_always_1l(get_field,32)
      call cm_u_subtract_constant_3_1l(i1,index_1_id,1,32)
      call cm_u_subtract_constant_3_1l(i2,index_2_id,1,32)
      call cm_deposit_news_coordinate_1l(source_geo, get_field,1,
     +                                   i1,32)
      call cm_deposit_news_coordinate_1l(source_geo, get_field,2,
     +                                   i2,32)
      call cm_get_1l(dest_temp_id,get_field,source_temp_id,2*32)
      call CM_u_move_1L(dest1_id, dest_temp_id, 32)
      call CM_u_move_1L(dest2_id, dest_temp_id + 32, 32)

      call cm_deallocate_stack_through(dest_temp_id)
      call CMF_set_is_modified(dest1,MODIF)
      call CMF_set_is_modified(dest2,MODIF)
      return
      end



      subroutine gather3_2d(dest1, dest2, dest3, index_1, index_2,
     +                      source1, source2, source3)

      include '/usr/include/cm/paris-configuration-fort.h'
      include '/usr/include/cm/CMF_defs.h'

c   compute  dest = source(index_1,index_2)
c   gets source (2-dimensional) for dest (n-dimensional)
c   the dest and index fields must be in the same vp set
c
c parameters
c    dest1      : real destination field; n-dimensional
c    dest2      : real destination field; n-dimensional
c    dest3      : real destination field; n-dimensional
c    index_1    : integer field ; first index of source array
c    index_2    : integer field ; second index of source array
c    source1    : real source field; 2-dimensional
c    source2    : real source field; 2-dimensional
```

```
c     source3     : real source field; 2-dimensional

      integer dest1, dest2, dest3
      integer source1, source2, source3
      integer dest1_id, dest2_id, dest3_id
      integer dest_vp_set,dest_geo
      integer index_1,index_1_id
      integer index_2,index_2_id
      integer source1_id, source2_id, source3_id
      integer source_vp_set,source_geo
      integer get_field,length,temp,i1,i2
      integer i,rank

      integer source_temp_id, dest_temp_id

      dest1_id = cmf_get_field_id(dest1)
      dest2_id = cmf_get_field_id(dest2)
      dest3_id = cmf_get_field_id(dest3)

      index_1_id = cmf_get_field_id(index_1)
      index_2_id = cmf_get_field_id(index_2)
      source1_id = cmf_get_field_id(source1)
      source2_id = cmf_get_field_id(source2)
      source3_id = cmf_get_field_id(source3)

      if (dest1_id .eq. 0) then
         print *,
     +'Error, the dest1 argument to gather3_2d  is not on the CM'
         stop
      endif

      if (dest2_id .eq. 0) then
         print *,
     +'Error, the dest2 argument to gather3_2d  is not on the CM'
         stop
      endif

      if (dest3_id .eq. 0) then
         print *,
     +'Error, the dest3 argument to gather3_2d  is not on the CM'
         stop
      endif

      if (index_1_id .eq. 0) then
         print *,
     +'Error, the index_1 argument to gather3_2d is not on the CM'
         stop
      endif
```

```
      if (index_2_id .eq. 0) then
         print *,
     +'Error, the index_2 argument to gather3_2d is not on the CM'
         stop
      endif

      if (source1_id .eq. 0) then
         print *,
     +'Error, the source1 argument to gather3_2d is not on the CM'
         stop
      endif

      if (source2_id .eq. 0) then
         print *,
     +'Error, the source2 argument to gather3_2d is not on the CM'
         stop
      endif

      if (source3_id .eq. 0) then
         print *,
     +'Error, the source3 argument to gather3_2d is not on the CM'
         stop
      endif

      dest_vp_set = cmf_get_vp_set_id(dest1)
      dest_geo = cm_vp_set_geometry(dest_vp_set)

      source_vp_set = cmf_get_vp_set_id(source1)
      source_geo = cm_vp_set_geometry(source_vp_set)
      call cm_set_vp_set(source_vp_set)
      source_temp_id = CM_allocate_stack_field(3*32)
      call CM_u_move_1L(source_temp_id, source1_id, 32)
      call CM_u_move_1L(source_temp_id + 32, source2_id, 32)
      call CM_u_move_1L(source_temp_id + 64, source3_id, 32)

      call cm_set_vp_set(dest_vp_set)

      get_field = cm_allocate_stack_field(32)
      temp = cm_allocate_stack_field(32)
      i1 = cm_allocate_stack_field(32)
      i2 = cm_allocate_stack_field(32)
      call cm_set_context()
      dest_temp_id = CM_allocate_stack_field(3*32)

c      select context for destination

      call cm_my_news_coordinate_1l(temp,0,32)
```

```
      call cm_u_eq_constant_1l(temp,0,32)
      call cm_logand_context_with_test()

      rank = cm_geometry_rank(dest_geo)
      do i=1,rank-1
         length = cmf_get_axis_extent(dest1,i-1)
         call CM_my_news_coordinate_1L(temp,i,32)
         call CM_u_lt_constant_1L(temp,length,32)
         call CM_logand_context_with_test()
      enddo

      call cm_u_move_zero_always_1l(get_field,32)
      call cm_u_subtract_constant_3_1l(i1,index_1_id,1,32)
      call cm_u_subtract_constant_3_1l(i2,index_2_id,1,32)
      call cm_deposit_news_coordinate_1l(source_geo, get_field,1,
     +                                   i1,32)
      call cm_deposit_news_coordinate_1l(source_geo, get_field,2,
     +                                   i2,32)
      call cm_get_1l(dest_temp_id,get_field,source_temp_id,3*32)
      call CM_u_move_1L(dest1_id, dest_temp_id, 32)
      call CM_u_move_1L(dest2_id, dest_temp_id + 32, 32)
      call CM_u_move_1L(dest3_id, dest_temp_id + 64, 32)

      call cm_deallocate_stack_through(dest_temp_id)
      call CMF_set_is_modified(dest1,MODIF)
      call CMF_set_is_modified(dest2,MODIF)
      call CMF_set_is_modified(dest3,MODIF)
      return
      end



      subroutine gather4_2d(dest1, dest2, dest3, dest4, index_1,
     +                      index_2, source1, source2, source3,
     +                      source4)

      include '/usr/include/cm/paris-configuration-fort.h'
      include '/usr/include/cm/CMF_defs.h'

c   compute  dest = source(index_1,index_2)
c   gets source (2-dimensional) for dest (n-dimensional)
c   the dest and index fields must be in the same vp set
c
c parameters
c    dest1      : real destination field; n-dimensional
c    dest2      : real destination field; n-dimensional
c    dest3      : real destination field; n-dimensional
c    dest4      : real destination field; n-dimensional
c    index_1    : integer field ; first index of source array
```

```
c     index_2     : integer field ; second index of source array
c     source1     : real source field; 2-dimensional
c     source2     : real source field; 2-dimensional
c     source3     : real source field; 2-dimensional
c     source4     : real source field; 2-dimensional

      integer dest1, dest2, dest3, dest4
      integer source1, source2, source3, source4
      integer dest1_id, dest2_id, dest3_id, dest4_id
      integer dest_vp_set,dest_geo
      integer index_1,index_1_id
      integer index_2,index_2_id
      integer source1_id, source2_id, source3_id, source4_id
      integer source_vp_set,source_geo
      integer get_field,length,temp,i1,i2
      integer i,rank

      integer source_temp_id, dest_temp_id

      dest1_id = cmf_get_field_id(dest1)
      dest2_id = cmf_get_field_id(dest2)
      dest3_id = cmf_get_field_id(dest3)
      dest4_id = cmf_get_field_id(dest4)

      index_1_id = cmf_get_field_id(index_1)
      index_2_id = cmf_get_field_id(index_2)
      source1_id = cmf_get_field_id(source1)
      source2_id = cmf_get_field_id(source2)
      source3_id = cmf_get_field_id(source3)
      source4_id = cmf_get_field_id(source4)

      if (dest1_id .eq. 0) then
         print *,
     +'Error, the dest1 argument to gather4_2d  is not on the CM'
         stop
      endif

      if (dest2_id .eq. 0) then
         print *,
     +'Error, the dest2 argument to gather4_2d  is not on the CM'
         stop
      endif

      if (dest3_id .eq. 0) then
         print *,
     +'Error, the dest3 argument to gather4_2d  is not on the CM'
         stop
      endif
```

```
 if (dest4_id .eq. 0) then
    print *,
+'Error, the dest4 argument to gather4_2d  is not on the CM'
    stop
 endif


 if (index_1_id .eq. 0) then
    print *,
+'Error, the index_1 argument to gather4_2d is not on the CM'
    stop
 endif

 if (index_2_id .eq. 0) then
    print *,
+'Error, the index_2 argument to gather4_2d is not on the CM'
    stop
 endif

 if (source1_id .eq. 0) then
    print *,
+'Error, the source1 argument to gather4_2d is not on the CM'
    stop
 endif

 if (source2_id .eq. 0) then
    print *,
+'Error, the source2 argument to gather4_2d is not on the CM'
    stop
 endif

 if (source3_id .eq. 0) then
    print *,
+'Error, the source3 argument to gather4_2d is not on the CM'
    stop
 endif

 if (source4_id .eq. 0) then
    print *,
+'Error, the source4 argument to gather4_2d is not on the CM'
    stop
 endif

 dest_vp_set = cmf_get_vp_set_id(dest1)
 dest_geo = cm_vp_set_geometry(dest_vp_set)

 source_vp_set = cmf_get_vp_set_id(source1)
```

```
      source_geo = cm_vp_set_geometry(source_vp_set)
      call cm_set_vp_set(source_vp_set)
      source_temp_id = CM_allocate_stack_field(4*32)
      call CM_u_move_1L(source_temp_id, source1_id, 32)
      call CM_u_move_1L(source_temp_id + 32, source2_id, 32)
      call CM_u_move_1L(source_temp_id + 64, source3_id, 32)
      call CM_u_move_1L(source_temp_id + 96, source4_id, 32)

      call cm_set_vp_set(dest_vp_set)

      get_field = cm_allocate_stack_field(32)
      temp = cm_allocate_stack_field(32)
      i1 = cm_allocate_stack_field(32)
      i2 = cm_allocate_stack_field(32)
      call cm_set_context()
      dest_temp_id = CM_allocate_stack_field(4*32)

c      select context for destination

      call cm_my_news_coordinate_1l(temp,0,32)
      call cm_u_eq_constant_1l(temp,0,32)
      call cm_logand_context_with_test()

      rank = cm_geometry_rank(dest_geo)
      do i=1,rank-1
         length = cmf_get_axis_extent(dest1,i-1)
         call CM_my_news_coordinate_1L(temp,i,32)
         call CM_u_lt_constant_1L(temp,length,32)
         call CM_logand_context_with_test()
      enddo

      call cm_u_move_zero_always_1l(get_field,32)
      call cm_u_subtract_constant_3_1l(i1,index_1_id,1,32)
      call cm_u_subtract_constant_3_1l(i2,index_2_id,1,32)
      call cm_deposit_news_coordinate_1l(source_geo, get_field,1,
     +                                   i1,32)
      call cm_deposit_news_coordinate_1l(source_geo, get_field,2,
     +                                   i2,32)
      call cm_get_1l(dest_temp_id,get_field,source_temp_id,4*32)
      call CM_u_move_1L(dest1_id, dest_temp_id, 32)
      call CM_u_move_1L(dest2_id, dest_temp_id + 32, 32)
      call CM_u_move_1L(dest3_id, dest_temp_id + 64, 32)
      call CM_u_move_1L(dest4_id, dest_temp_id + 96, 32)

      call cm_deallocate_stack_through(dest_temp_id)
      call CMF_set_is_modified(dest1,MODIF)
      call CMF_set_is_modified(dest2,MODIF)
      call CMF_set_is_modified(dest3,MODIF)
```

```
      call CMF_set_is_modified(dest4,MODIF)
      return
      end



      subroutine gather1_3d(dest,index_1,index_2,index_3,source)

      include '/usr/include/cm/paris-configuration-fort.h'
      include '/usr/include/cm/CMF_defs.h'

c   compute  dest = source(index_1,index_2,index_3)
c   gets source (3-dimensional) for dest (n-dimensional)
c   the dest and index fields must be in the same vp set
c
c parameters
c    dest      : real destination field; n-dimensional
c    index_1   : integer field ; first index of source array
c    index_2   : integer field ; second index of source array
c    index_3   : integer field ; third index of source array
c    source    : real source field; 3-dimensional

      integer dest,dest_id,dest_vp_set,dest_geo
      integer index_1,index_1_id
      integer index_2,index_2_id
      integer index_3,index_3_id
      integer source,source_id,source_vp_set,source_geo
      integer get_field,length,temp,i1,i2,i3
      integer i,rank

      dest_id = cmf_get_field_id(dest)
      index_1_id = cmf_get_field_id(index_1)
      index_2_id = cmf_get_field_id(index_2)
      index_3_id = cmf_get_field_id(index_3)
      source_id = cmf_get_field_id(source)

      if (dest_id .eq. 0) then
         print *,
     +'Error, the dest argument to gather1_3d  is not on the CM'
         stop
      endif

      if (index_1_id .eq. 0) then
         print *,
     +'Error, the index_1 argument to gather1_3d is not on the CM'
         stop
      endif

      if (index_2_id .eq. 0) then
```

```
      print *,
     +'Error, the index_2 argument to gather1_3d is not on the CM'
      stop
      endif

      if (index_3_id .eq. 0) then
      print *,
     +'Error, the index_3 argument to gather1_3d is not on the CM'
      stop
      endif

      if (source_id .eq. 0) then
      print *,
     +'Error, the source argument to gather1_3d is not on the CM'
      stop
      endif

      dest_vp_set = cmf_get_vp_set_id(dest)
      dest_geo = cm_vp_set_geometry(dest_vp_set)

      source_vp_set = cmf_get_vp_set_id(source)
      source_geo = cm_vp_set_geometry(source_vp_set)

      call cm_set_vp_set(dest_vp_set)

      get_field = cm_allocate_stack_field(32)
      temp = cm_allocate_stack_field(32)
      i1 = cm_allocate_stack_field(32)
      i2 = cm_allocate_stack_field(32)
      i3 = cm_allocate_stack_field(32)

c      select context for destination

      call cm_set_context()
      call cm_my_news_coordinate_1l(temp,0,32)
      call cm_u_eq_constant_1l(temp,0,32)
      call cm_logand_context_with_test()

      rank = cm_geometry_rank(dest_geo)
      do i=1,rank-1
      length = cmf_get_axis_extent(dest,i-1)
      call CM_my_news_coordinate_1L(temp,i,32)
      call CM_u_lt_constant_1L(temp,length,32)
      call CM_logand_context_with_test()
      enddo

      call cm_u_move_zero_always_1l(get_field,32)
      call cm_u_subtract_constant_3_1l(i1,index_1_id,1,32)
```

```
      call cm_u_subtract_constant_3_1l(i2,index_2_id,1,32)
      call cm_u_subtract_constant_3_1l(i3,index_3_id,1,32)
      call cm_deposit_news_coordinate_1l(source_geo, get_field,1,
     +                                   i1,32)
      call cm_deposit_news_coordinate_1l(source_geo, get_field,2,
     +                                   i2,32)
      call cm_deposit_news_coordinate_1l(source_geo, get_field,3,
     +                                   i3,32)
      call cm_get_1l(dest_id,get_field,source_id,32)

      call cm_deallocate_stack_through(get_field)
      call CMF_set_is_modified(dest,MODIF)
      return
      end



      subroutine gather2_3d(dest1, dest2, index_1, index_2, index_3,
     +                      source1, source2)

      include '/usr/include/cm/paris-configuration-fort.h'
      include '/usr/include/cm/CMF_defs.h'

c   compute  dest1 = source1(index_1,index_2,index_3)
c            dest2 = source2(index_1,index_2,index_3)
c
c   gets source (3-dimensional) for dest (n-dimensional)
c   the dest and index fields must be in the same vp set
c
c parameters
c    dest1      : real destination field; n-dimensional
c    dest2      : real destination field; n-dimensional
c    index_1    : integer field ; first index of source array
c    index_2    : integer field ; second index of source array
c    index_3    : integer field ; third index of source array
c    source1    : real source field; 3-dimensional
c    source2    : real source field; 3-dimensional

      integer dest1, dest2
      integer source1, source2
      integer dest1_id, dest2_id
      integer dest_vp_set,dest_geo
      integer index_1,index_1_id
      integer index_2,index_2_id
      integer index_3,index_3_id
      integer source1_id, source2_id
      integer source_vp_set,source_geo
      integer get_field,length,temp,i1,i2,i3
      integer i,rank
```

```
      integer source_temp_id, dest_temp_id

      dest1_id = cmf_get_field_id(dest1)
      dest2_id = cmf_get_field_id(dest2)

      index_1_id = cmf_get_field_id(index_1)
      index_2_id = cmf_get_field_id(index_2)
      index_3_id = cmf_get_field_id(index_3)
      source1_id = cmf_get_field_id(source1)
      source2_id = cmf_get_field_id(source2)

      if (dest1_id .eq. 0) then
         print *,
     +'Error, the dest1 argument to gather2_3d  is not on the CM'
         stop
      endif

      if (dest2_id .eq. 0) then
         print *,
     +'Error, the dest2 argument to gather2_3d  is not on the CM'
         stop
      endif

      if (index_1_id .eq. 0) then
         print *,
     +'Error, the index_1 argument to gather2_3d is not on the CM'
         stop
      endif

      if (index_2_id .eq. 0) then
         print *,
     +'Error, the index_2 argument to gather2_3d is not on the CM'
         stop
      endif

      if (index_3_id .eq. 0) then
         print *,
     +'Error, the index_3 argument to gather2_3d is not on the CM'
         stop
      endif

      if (source1_id .eq. 0) then
         print *,
     +'Error, the source1 argument to gather2_3d is not on the CM'
         stop
      endif
```

```
      if (source2_id .eq. 0) then
         print *,
     +'Error, the source2 argument to gather2_3d is not on the CM'
         stop
      endif

      dest_vp_set = cmf_get_vp_set_id(dest1)
      dest_geo = cm_vp_set_geometry(dest_vp_set)

      source_vp_set = cmf_get_vp_set_id(source1)
      source_geo = cm_vp_set_geometry(source_vp_set)
      call cm_set_vp_set(source_vp_set)
      source_temp_id = CM_allocate_stack_field(2*32)
      call CM_u_move_1L(source_temp_id, source1_id, 32)
      call CM_u_move_1L(source_temp_id + 32, source2_id, 32)

      call cm_set_vp_set(dest_vp_set)

      get_field = cm_allocate_stack_field(32)
      temp = cm_allocate_stack_field(32)
      i1 = cm_allocate_stack_field(32)
      i2 = cm_allocate_stack_field(32)
      i3 = cm_allocate_stack_field(32)
      call cm_set_context()
      dest_temp_id = CM_allocate_stack_field(2*32)

c      select context for destination

      call cm_my_news_coordinate_1l(temp,0,32)
      call cm_u_eq_constant_1l(temp,0,32)
      call cm_logand_context_with_test()

      rank = cm_geometry_rank(dest_geo)
      do i=1,rank-1
         length = cmf_get_axis_extent(dest1,i-1)
         call CM_my_news_coordinate_1L(temp,i,32)
         call CM_u_lt_constant_1L(temp,length,32)
         call CM_logand_context_with_test()
      enddo

      call cm_u_move_zero_always_1l(get_field,32)
      call cm_u_subtract_constant_3_1l(i1,index_1_id,1,32)
      call cm_u_subtract_constant_3_1l(i2,index_2_id,1,32)
      call cm_u_subtract_constant_3_1l(i3,index_3_id,1,32)
      call cm_deposit_news_coordinate_1l(source_geo, get_field,1,
     +                                   i1,32)
      call cm_deposit_news_coordinate_1l(source_geo, get_field,2,
     +                                   i2,32)
```

```
      call cm_deposit_news_coordinate_1l(source_geo, get_field,3,
     +                                   i3,32)
      call cm_get_1l(dest_temp_id,get_field,source_temp_id,2*32)
      call CM_u_move_1L(dest1_id, dest_temp_id, 32)
      call CM_u_move_1L(dest2_id, dest_temp_id + 32, 32)

      call cm_deallocate_stack_through(dest_temp_id)
      call CMF_set_is_modified(dest1,MODIF)
      call CMF_set_is_modified(dest2,MODIF)
      return
      end



      subroutine gather3_3d(dest1, dest2, dest3, index_1, index_2,
     +                      index_3, source1, source2, source3)

      include '/usr/include/cm/paris-configuration-fort.h'
      include '/usr/include/cm/CMF_defs.h'

c   compute  dest = source(index_1,index_2,index_3)
c   gets source (3-dimensional) for dest (n-dimensional)
c   the dest and index fields must be in the same vp set
c
c parameters
c    dest1      : real destination field; n-dimensional
c    dest2      : real destination field; n-dimensional
c    dest3      : real destination field; n-dimensional
c    index_1    : integer field ; first index of source array
c    index_2    : integer field ; second index of source array
c    index_3    : integer field ; third index of source array
c    source1    : real source field; 3-dimensional
c    source2    : real source field; 3-dimensional
c    source3    : real source field; 3-dimensional

      integer dest1, dest2, dest3
      integer source1, source2, source3
      integer dest1_id, dest2_id, dest3_id
      integer dest_vp_set,dest_geo
      integer index_1,index_1_id
      integer index_2,index_2_id
      integer index_3,index_3_id
      integer source1_id, source2_id, source3_id
      integer source_vp_set,source_geo
      integer get_field,length,temp,i1,i2,i3
      integer i,rank

      integer source_temp_id, dest_temp_id
```

```
 dest1_id = cmf_get_field_id(dest1)
 dest2_id = cmf_get_field_id(dest2)
 dest3_id = cmf_get_field_id(dest3)

 index_1_id = cmf_get_field_id(index_1)
 index_2_id = cmf_get_field_id(index_2)
 index_3_id = cmf_get_field_id(index_3)
 source1_id = cmf_get_field_id(source1)
 source2_id = cmf_get_field_id(source2)
 source3_id = cmf_get_field_id(source3)

 if (dest1_id .eq. 0) then
    print *,
+'Error, the dest1 argument to gather3_3d  is not on the CM'
    stop
 endif

 if (dest2_id .eq. 0) then
    print *,
+'Error, the dest2 argument to gather3_3d  is not on the CM'
    stop
 endif

 if (dest3_id .eq. 0) then
    print *,
+'Error, the dest3 argument to gather3_3d  is not on the CM'
    stop
 endif

 if (index_1_id .eq. 0) then
    print *,
+'Error, the index_1 argument to gather3_3d is not on the CM'
    stop
 endif

 if (index_2_id .eq. 0) then
    print *,
+'Error, the index_2 argument to gather3_3d is not on the CM'
    stop
 endif

 if (index_3_id .eq. 0) then
    print *,
+'Error, the index_3 argument to gather3_3d is not on the CM'
    stop
 endif

 if (source1_id .eq. 0) then
```

```
      print *,
     +'Error, the source1 argument to gather3_3d is not on the CM'
      stop
     endif

     if (source2_id .eq. 0) then
      print *,
     +'Error, the source2 argument to gather3_3d is not on the CM'
      stop
     endif

     if (source3_id .eq. 0) then
      print *,
     +'Error, the source3 argument to gather3_3d is not on the CM'
      stop
     endif

     dest_vp_set = cmf_get_vp_set_id(dest1)
     dest_geo = cm_vp_set_geometry(dest_vp_set)

     source_vp_set = cmf_get_vp_set_id(source1)
     source_geo = cm_vp_set_geometry(source_vp_set)
     call cm_set_vp_set(source_vp_set)
     source_temp_id = CM_allocate_stack_field(3*32)
     call CM_u_move_1L(source_temp_id, source1_id, 32)
     call CM_u_move_1L(source_temp_id + 32, source2_id, 32)
     call CM_u_move_1L(source_temp_id + 64, source3_id, 32)

     call cm_set_vp_set(dest_vp_set)

     get_field = cm_allocate_stack_field(32)
     temp = cm_allocate_stack_field(32)
     i1 = cm_allocate_stack_field(32)
     i2 = cm_allocate_stack_field(32)
     i3 = cm_allocate_stack_field(32)
     call cm_set_context()
     dest_temp_id = CM_allocate_stack_field(3*32)

c     select context for destination

     call cm_my_news_coordinate_1l(temp,0,32)
     call cm_u_eq_constant_1l(temp,0,32)
     call cm_logand_context_with_test()

     rank = cm_geometry_rank(dest_geo)
     do i=1,rank-1
      length = cmf_get_axis_extent(desti,i-1)
      call CM_my_news_coordinate_1L(temp,i,32)
```

```
         call CM_u_lt_constant_1L(temp,length,32)
         call CM_logand_context_with_test()
       enddo

       call cm_u_move_zero_always_1l(get_field,32)
       call cm_u_subtract_constant_3_1l(i1,index_1_id,1,32)
       call cm_u_subtract_constant_3_1l(i2,index_2_id,1,32)
       call cm_u_subtract_constant_3_1l(i3,index_3_id,1,32)
       call cm_deposit_news_coordinate_1l(source_geo, get_field,1,
      +                                   i1,32)
       call cm_deposit_news_coordinate_1l(source_geo, get_field,2,
      +                                   i2,32)
       call cm_deposit_news_coordinate_1l(source_geo, get_field,3,
      +                                   i3,32)
       call cm_get_1l(dest_temp_id,get_field,source_temp_id,3*32)
       call CM_u_move_1L(dest1_id, dest_temp_id, 32)
       call CM_u_move_1L(dest2_id, dest_temp_id + 32, 32)
       call CM_u_move_1L(dest3_id, dest_temp_id + 64, 32)

       call cm_deallocate_stack_through(dest_temp_id)
       call CMF_set_is_modified(dest1,MODIF)
       call CMF_set_is_modified(dest2,MODIF)
       call CMF_set_is_modified(dest3,MODIF)
       return
       end



       subroutine gather4_3d(dest1, dest2, dest3, dest4, index_1,
      +                      index_2, index_3, source1, source2,
      +                      source3, source4)

       include '/usr/include/cm/paris-configuration-fort.h'
       include '/usr/include/cm/CMF_defs.h'

c    compute  dest = source(index_1,index_2,index_3)
c    gets source (3-dimensional) for dest (n-dimensional)
c    the dest and index fields must be in the same vp set
c
c parameters
c     dest1      : real destination field; n-dimensional
c     dest2      : real destination field; n-dimensional
c     dest3      : real destination field; n-dimensional
c     dest4      : real destination field; n-dimensional
c     index_1    : integer field ; first index of source array
c     index_2    : integer field ; second index of source array
c     source1    : real source field; 3-dimensional
c     source2    : real source field; 3-dimensional
c     source3    : real source field; 3-dimensional
```

```
c     source4     : real source field; 3-dimensional

      integer dest1, dest2, dest3, dest4
      integer source1, source2, source3, source4
      integer dest1_id, dest2_id, dest3_id, dest4_id
      integer dest_vp_set,dest_geo
      integer index_1,index_1_id
      integer index_2,index_2_id
      integer index_3,index_3_id
      integer source1_id, source2_id, source3_id, source4_id
      integer source_vp_set,source_geo
      integer get_field,length,temp,i1,i2,i3
      integer i,rank

      integer source_temp_id, dest_temp_id

      dest1_id = cmf_get_field_id(dest1)
      dest2_id = cmf_get_field_id(dest2)
      dest3_id = cmf_get_field_id(dest3)
      dest4_id = cmf_get_field_id(dest4)

      index_1_id = cmf_get_field_id(index_1)
      index_2_id = cmf_get_field_id(index_2)
      index_3_id = cmf_get_field_id(index_3)
      source1_id = cmf_get_field_id(source1)
      source2_id = cmf_get_field_id(source2)
      source3_id = cmf_get_field_id(source3)
      source4_id = cmf_get_field_id(source4)

      if (dest1_id .eq. 0) then
         print *,
     +'Error, the dest1 argument to gather4_3d  is not on the CM'
         stop
      endif

      if (dest2_id .eq. 0) then
         print *,
     +'Error, the dest2 argument to gather4_3d  is not on the CM'
         stop
      endif

      if (dest3_id .eq. 0) then
         print *,
     +'Error, the dest3 argument to gather4_3d  is not on the CM'
         stop
      endif

      if (dest4_id .eq. 0) then
```

```
      print *,
     +'Error, the dest4 argument to gather4_3d  is not on the CM'
      stop
     endif


     if (index_1_id .eq. 0) then
      print *,
     +'Error, the index_1 argument to gather4_3d is not on the CM'
      stop
     endif

     if (index_2_id .eq. 0) then
      print *,
     +'Error, the index_2 argument to gather4_3d is not on the CM'
      stop
     endif

     if (index_3_id .eq. 0) then
      print *,
     +'Error, the index_3 argument to gather4_3d is not on the CM'
      stop
     endif

     if (source1_id .eq. 0) then
      print *,
     +'Error, the source1 argument to gather4_3d is not on the CM'
      stop
     endif

     if (source2_id .eq. 0) then
      print *,
     +'Error, the source2 argument to gather4_3d is not on the CM'
      stop
     endif

     if (source3_id .eq. 0) then
      print *,
     +'Error, the source3 argument to gather4_3d is not on the CM'
      stop
     endif

     if (source4_id .eq. 0) then
      print *,
     +'Error, the source4 argument to gather4_3d is not on the CM'
      stop
     endif
```

```
      dest_vp_set = cmf_get_vp_set_id(dest1)
      dest_geo = cm_vp_set_geometry(dest_vp_set)

      source_vp_set = cmf_get_vp_set_id(source1)
      source_geo = cm_vp_set_geometry(source_vp_set)
      call cm_set_vp_set(source_vp_set)
      source_temp_id = CM_allocate_stack_field(4*32)
      call CM_u_move_1L(source_temp_id, source1_id, 32)
      call CM_u_move_1L(source_temp_id + 32, source2_id, 32)
      call CM_u_move_1L(source_temp_id + 64, source3_id, 32)
      call CM_u_move_1L(source_temp_id + 96, source4_id, 32)

      call cm_set_vp_set(dest_vp_set)

      get_field = cm_allocate_stack_field(32)
      temp = cm_allocate_stack_field(32)
      i1 = cm_allocate_stack_field(32)
      i2 = cm_allocate_stack_field(32)
      i3 = cm_allocate_stack_field(32)
      call cm_set_context()
      dest_temp_id = CM_allocate_stack_field(4*32)

c      select context for destination

      call cm_my_news_coordinate_1l(temp,0,32)
      call cm_u_eq_constant_1l(temp,0,32)
      call cm_logand_context_with_test()

      rank = cm_geometry_rank(dest_geo)
      do i=1,rank-1
         length = cmf_get_axis_extent(dest1,i-1)
         call CM_my_news_coordinate_1L(temp,i,32)
         call CM_u_lt_constant_1L(temp,length,32)
         call CM_logand_context_with_test()
      enddo

      call cm_u_move_zero_always_1l(get_field,32)
      call cm_u_subtract_constant_3_1l(i1,index_1_id,1,32)
      call cm_u_subtract_constant_3_1l(i2,index_2_id,1,32)
      call cm_u_subtract_constant_3_1l(i3,index_3_id,1,32)
      call cm_deposit_news_coordinate_1l(source_geo, get_field,1,
     +                                   i1,32)
      call cm_deposit_news_coordinate_1l(source_geo, get_field,2,
     +                                   i2,32)
      call cm_deposit_news_coordinate_1l(source_geo, get_field,3,
     +                                   i3,32)
      call cm_get_1l(dest_temp_id,get_field,source_temp_id,4*32)
      call CM_u_move_1L(dest1_id, dest_temp_id, 32)
```

```
      call CM_u_move_1L(dest2_id, dest_temp_id + 32, 32)
      call CM_u_move_1L(dest3_id, dest_temp_id + 64, 32)
      call CM_u_move_1L(dest4_id, dest_temp_id + 96, 32)

      call cm_deallocate_stack_through(dest_temp_id)
      call CMF_set_is_modified(dest1,MODIF)
      call CMF_set_is_modified(dest2,MODIF)
      call CMF_set_is_modified(dest3,MODIF)
      call CMF_set_is_modified(dest4,MODIF)
      return
      end



      subroutine gather1_4d(dest,index_1,index_2,index_3,index_4,
     .                      source)

      include '/usr/include/cm/paris-configuration-fort.h'
      include '/usr/include/cm/CMF_defs.h'

c   compute  dest = source(index_1,index_2,index_3,index_4)
c   gets source (4-dimensional) for dest (n-dimensional)
c   the dest and index fields must be in the same vp set
c
c parameters
c    dest      : real destination field; n-dimensional
c    index_1   : integer field ; first index of source array
c    index_2   : integer field ; second index of source array
c    index_3   : integer field ; third index of source array
c    index_4   : integer field ; fourth index of source array
c    source    : real source field; 3-dimensional

      integer dest,dest_id,dest_vp_set,dest_geo
      integer index_1,index_1_id
      integer index_2,index_2_id
      integer index_3,index_3_id
      integer index_4,index_4_id
      integer source,source_id,source_vp_set,source_geo
      integer get_field,length,temp,i1,i2,i3,i4
      integer i,rank

      dest_id = cmf_get_field_id(dest)
      index_1_id = cmf_get_field_id(index_1)
      index_2_id = cmf_get_field_id(index_2)
      index_3_id = cmf_get_field_id(index_3)
      index_4_id = cmf_get_field_id(index_4)
      source_id = cmf_get_field_id(source)

      if (dest_id .eq. 0) then
```

```
      print *,
     +'Error, the dest argument to gather1_4d  is not on the CM'
      stop
     endif

     if (index_1_id .eq. 0) then
      print *,
     +'Error, the index_1 argument to gather1_4d is not on the CM'
      stop
     endif

     if (index_2_id .eq. 0) then
      print *,
     +'Error, the index_2 argument to gather1_4d is not on the CM'
      stop
     endif

     if (index_3_id .eq. 0) then
      print *,
     +'Error, the index_3 argument to gather1_4d is not on the CM'
      stop
     endif

     if (index_4_id .eq. 0) then
      print *,
     +'Error, the index_4 argument to gather1_4d is not on the CM'
      stop
     endif

     if (source_id .eq. 0) then
      print *,
     +'Error, the source argument to gather1_4d is not on the CM'
      stop
     endif

     dest_vp_set = cmf_get_vp_set_id(dest)
     dest_geo = cm_vp_set_geometry(dest_vp_set)

     source_vp_set = cmf_get_vp_set_id(source)
     source_geo = cm_vp_set_geometry(source_vp_set)

     call cm_set_vp_set(dest_vp_set)

     get_field = cm_allocate_stack_field(32)
     temp = cm_allocate_stack_field(32)
     i1 = cm_allocate_stack_field(32)
     i2 = cm_allocate_stack_field(32)
     i3 = cm_allocate_stack_field(32)
```

```
      i4 = cm_allocate_stack_field(32)

c      select context for destination

      call cm_set_context()
      call cm_my_news_coordinate_1l(temp,0,32)
      call cm_u_eq_constant_1l(temp,0,32)
      call cm_logand_context_with_test()

      rank = cm_geometry_rank(dest_geo)
      do i=1,rank-1
         length = cmf_get_axis_extent(dest,i-1)
         call CM_my_news_coordinate_1L(temp,i,32)
         call CM_u_lt_constant_1L(temp,length,32)
         call CM_logand_context_with_test()
      enddo

      call cm_u_move_zero_always_1l(get_field,32)
      call cm_u_subtract_constant_3_1l(i1,index_1_id,1,32)
      call cm_u_subtract_constant_3_1l(i2,index_2_id,1,32)
      call cm_u_subtract_constant_3_1l(i3,index_3_id,1,32)
      call cm_u_subtract_constant_3_1l(i4,index_4_id,1,32)
      call cm_deposit_news_coordinate_1l(source_geo, get_field,1,
     +                                   i1,32)
      call cm_deposit_news_coordinate_1l(source_geo, get_field,2,
     +                                   i2,32)
      call cm_deposit_news_coordinate_1l(source_geo, get_field,3,
     +                                   i3,32)
      call cm_deposit_news_coordinate_1l(source_geo, get_field,4,
     +                                   i4,32)
      call cm_get_1l(dest_id,get_field,source_id,32)

      call cm_deallocate_stack_through(get_field)
      call CMF_set_is_modified(dest,MODIF)
      return
      end



      subroutine gather2_4d(dest1, dest2, index_1, index_2, index_3,
     +                      index_4, source1, source2)

      include '/usr/include/cm/paris-configuration-fort.h'
      include '/usr/include/cm/CMF_defs.h'

c   compute  dest1 = source1(index_1,index_2,index_3,index_4)
c            dest2 = source2(index_1,index_2,index_3,index_4)
c
c   gets source (4-dimensional) for dest (n-dimensional)
```

```
c   the dest and index fields must be in the same vp set
c
c parameters
c    dest1      : real destination field; n-dimensional
c    dest2      : real destination field; n-dimensional
c    index_1    : integer field ; first index of source array
c    index_2    : integer field ; second index of source array
c    index_3    : integer field ; third index of source array
c    index_4    : integer field ; fourth index of source array
c    source1    : real source field; 3-dimensional
c    source2    : real source field; 3-dimensional

      integer dest1, dest2
      integer source1, source2
      integer dest1_id, dest2_id
      integer dest_vp_set,dest_geo
      integer index_1,index_1_id
      integer index_2,index_2_id
      integer index_3,index_3_id
      integer index_4,index_4_id
      integer source1_id, source2_id
      integer source_vp_set,source_geo
      integer get_field,length,temp,i1,i2,i3,i4
      integer i,rank

      integer source_temp_id, dest_temp_id

      dest1_id = cmf_get_field_id(dest1)
      dest2_id = cmf_get_field_id(dest2)

      index_1_id = cmf_get_field_id(index_1)
      index_2_id = cmf_get_field_id(index_2)
      index_3_id = cmf_get_field_id(index_3)
      index_4_id = cmf_get_field_id(index_4)
      source1_id = cmf_get_field_id(source1)
      source2_id = cmf_get_field_id(source2)

      if (dest1_id .eq. 0) then
         print *,
     +'Error, the dest1 argument to gather2_4d  is not on the CM'
         stop
      endif

      if (dest2_id .eq. 0) then
         print *,
     +'Error, the dest2 argument to gather2_4d  is not on the CM'
         stop
      endif
```

```
 if (index_1_id .eq. 0) then
    print *,
+'Error, the index_1 argument to gather2_4d is not on the CM'
    stop
 endif

 if (index_2_id .eq. 0) then
    print *,
+'Error, the index_2 argument to gather2_4d is not on the CM'
    stop
 endif

 if (index_3_id .eq. 0) then
    print *,
+'Error, the index_3 argument to gather2_4d is not on the CM'
    stop
 endif

 if (index_4_id .eq. 0) then
    print *,
+'Error, the index_4 argument to gather2_4d is not on the CM'
    stop
 endif

 if (source1_id .eq. 0) then
    print *,
+'Error, the source1 argument to gather2_4d is not on the CM'
    stop
 endif

 if (source2_id .eq. 0) then
    print *,
+'Error, the source2 argument to gather2_4d is not on the CM'
    stop
 endif

 dest_vp_set = cmf_get_vp_set_id(dest1)
 dest_geo = cm_vp_set_geometry(dest_vp_set)

 source_vp_set = cmf_get_vp_set_id(source1)
 source_geo = cm_vp_set_geometry(source_vp_set)
 call cm_set_vp_set(source_vp_set)
 source_temp_id = CM_allocate_stack_field(2*32)
 call CM_u_move_1L(source_temp_id, source1_id, 32)
 call CM_u_move_1L(source_temp_id + 32, source2_id, 32)

 call cm_set_vp_set(dest_vp_set)
```

```
      get_field = cm_allocate_stack_field(32)
      temp = cm_allocate_stack_field(32)
      i1 = cm_allocate_stack_field(32)
      i2 = cm_allocate_stack_field(32)
      i3 = cm_allocate_stack_field(32)
      i4 = cm_allocate_stack_field(32)
      call cm_set_context()
      dest_temp_id = CM_allocate_stack_field(2*32)

c      select context for destination

      call cm_my_news_coordinate_1l(temp,0,32)
      call cm_u_eq_constant_1l(temp,0,32)
      call cm_logand_context_with_test()

      rank = cm_geometry_rank(dest_geo)
      do i=1,rank-1
         length = cmf_get_axis_extent(dest1,i-1)
         call CM_my_news_coordinate_1L(temp,i,32)
         call CM_u_lt_constant_1L(temp,length,32)
         call CM_logand_context_with_test()
      enddo

      call cm_u_move_zero_always_1l(get_field,32)
      call cm_u_subtract_constant_3_1l(i1,index_1_id,1,32)
      call cm_u_subtract_constant_3_1l(i2,index_2_id,1,32)
      call cm_u_subtract_constant_3_1l(i3,index_3_id,1,32)
      call cm_u_subtract_constant_3_1l(i4,index_4_id,1,32)
      call cm_deposit_news_coordinate_1l(source_geo, get_field,1,
     +                                   i1,32)
      call cm_deposit_news_coordinate_1l(source_geo, get_field,2,
     +                                   i2,32)
      call cm_deposit_news_coordinate_1l(source_geo, get_field,3,
     +                                   i3,32)
      call cm_deposit_news_coordinate_1l(source_geo, get_field,4,
     +                                   i4,32)
      call cm_get_1l(dest_temp_id,get_field,source_temp_id,2*32)
      call CM_u_move_1L(dest1_id, dest_temp_id, 32)
      call CM_u_move_1L(dest2_id, dest_temp_id + 32, 32)

      call cm_deallocate_stack_through(dest_temp_id)
      call CMF_set_is_modified(dest1,MODIF)
      call CMF_set_is_modified(dest2,MODIF)
      return
      end
```

```
      subroutine gather3_4d(dest1, dest2, dest3, index_1, index_2,
     +                      index_3, index_4,source1, source2, source3)

      include '/usr/include/cm/paris-configuration-fort.h'
      include '/usr/include/cm/CMF_defs.h'

c   compute  dest = source(index_1,index_2,index_3,index_4)
c   gets source (4-dimensional) for dest (n-dimensional)
c   the dest and index fields must be in the same vp set
c
c parameters
c    dest1      : real destination field; n-dimensional
c    dest2      : real destination field; n-dimensional
c    dest3      : real destination field; n-dimensional
c    index_1    : integer field ; first index of source array
c    index_2    : integer field ; second index of source array
c    index_3    : integer field ; third index of source array
c    index_4    : integer field ; fourth index of source array
c    source1    : real source field; 3-dimensional
c    source2    : real source field; 3-dimensional
c    source3    : real source field; 3-dimensional

      integer dest1, dest2, dest3
      integer source1, source2, source3
      integer dest1_id, dest2_id, dest3_id
      integer dest_vp_set,dest_geo
      integer index_1,index_1_id
      integer index_2,index_2_id
      integer index_3,index_3_id
      integer index_4,index_4_id
      integer source1_id, source2_id, source3_id
      integer source_vp_set,source_geo
      integer get_field,length,temp,i1,i2,i3,i4
      integer i,rank

      integer source_temp_id, dest_temp_id

      dest1_id = cmf_get_field_id(dest1)
      dest2_id = cmf_get_field_id(dest2)
      dest3_id = cmf_get_field_id(dest3)

      index_1_id = cmf_get_field_id(index_1)
      index_2_id = cmf_get_field_id(index_2)
      index_3_id = cmf_get_field_id(index_3)
      index_4_id = cmf_get_field_id(index_4)
      source1_id = cmf_get_field_id(source1)
      source2_id = cmf_get_field_id(source2)
      source3_id = cmf_get_field_id(source3)
```

```
 if (dest1_id .eq. 0) then
    print *,
+'Error, the dest1 argument to gather3_4d  is not on the CM'
    stop
 endif

 if (dest2_id .eq. 0) then
    print *,
+'Error, the dest2 argument to gather3_4d  is not on the CM'
    stop
 endif

 if (dest3_id .eq. 0) then
    print *,
+'Error, the dest3 argument to gather3_4d  is not on the CM'
    stop
 endif

 if (index_1_id .eq. 0) then
    print *,
+'Error, the index_1 argument to gather3_4d is not on the CM'
    stop
 endif

 if (index_2_id .eq. 0) then
    print *,
+'Error, the index_2 argument to gather3_4d is not on the CM'
    stop
 endif

 if (index_3_id .eq. 0) then
    print *,
+'Error, the index_3 argument to gather3_4d is not on the CM'
    stop
 endif

 if (index_4_id .eq. 0) then
    print *,
+'Error, the index_4 argument to gather3_4d is not on the CM'
    stop
 endif

 if (source1_id .eq. 0) then
    print *,
+'Error, the source1 argument to gather3_4d is not on the CM'
    stop
 endif
```

```
      if (source2_id .eq. 0) then
         print *,
     +'Error, the source2 argument to gather3_4d is not on the CM'
         stop
      endif

      if (source3_id .eq. 0) then
         print *,
     +'Error, the source3 argument to gather3_4d is not on the CM'
         stop
      endif

      dest_vp_set = cmf_get_vp_set_id(dest1)
      dest_geo = cm_vp_set_geometry(dest_vp_set)

      source_vp_set = cmf_get_vp_set_id(source1)
      source_geo = cm_vp_set_geometry(source_vp_set)
      call cm_set_vp_set(source_vp_set)
      source_temp_id = CM_allocate_stack_field(3*32)
      call CM_u_move_1L(source_temp_id, source1_id, 32)
      call CM_u_move_1L(source_temp_id + 32, source2_id, 32)
      call CM_u_move_1L(source_temp_id + 64, source3_id, 32)

      call cm_set_vp_set(dest_vp_set)

      get_field = cm_allocate_stack_field(32)
      temp = cm_allocate_stack_field(32)
      i1 = cm_allocate_stack_field(32)
      i2 = cm_allocate_stack_field(32)
      i3 = cm_allocate_stack_field(32)
      i4 = cm_allocate_stack_field(32)
      call cm_set_context()
      dest_temp_id = CM_allocate_stack_field(3*32)

c      select context for destination

      call cm_my_news_coordinate_1l(temp,0,32)
      call cm_u_eq_constant_1l(temp,0,32)
      call cm_logand_context_with_test()

      rank = cm_geometry_rank(dest_geo)
      do i=1,rank-1
         length = cmf_get_axis_extent(dest1,i-1)
         call CM_my_news_coordinate_1L(temp,i,32)
         call CM_u_lt_constant_1L(temp,length,32)
         call CM_logand_context_with_test()
      enddo
```

```
      call cm_u_move_zero_always_1l(get_field,32)
      call cm_u_subtract_constant_3_1l(i1,index_1_id,1,32)
      call cm_u_subtract_constant_3_1l(i2,index_2_id,1,32)
      call cm_u_subtract_constant_3_1l(i3,index_3_id,1,32)
      call cm_u_subtract_constant_3_1l(i4,index_4_id,1,32)
      call cm_deposit_news_coordinate_1l(source_geo, get_field,1,
     +                                   i1,32)
      call cm_deposit_news_coordinate_1l(source_geo, get_field,2,
     +                                   i2,32)
      call cm_deposit_news_coordinate_1l(source_geo, get_field,3,
     +                                   i3,32)
      call cm_deposit_news_coordinate_1l(source_geo, get_field,4,
     +                                   i4,32)
      call cm_get_1l(dest_temp_id,get field,source temp_id,3*32)
      call CM_u_move_1L(dest1_id, dest_temp_id, 32)
      call CM_u_move_1L(dest2_id, dest_temp_id + 32, 32)
      call CM_u_move_1L(dest3_id, dest_temp_id + 64, 32)

      call cm_deallocate_stack_through(dest_temp_id)
      call CMF_set_is_modified(dest1,MODIF)
      call CMF_set_is_modified(dest2,MODIF)
      call CMF_set_is_modified(dest3,MODIF)
      return
      end



      subroutine gather4_4d(dest1, dest2, dest3, dest4, index_1,
     +                      index_2, index_3, index_4, source1,
     +                      source2, source3, source4)

      include '/usr/include/cm/paris-configuration-fort.h'
      include '/usr/include/cm/CMF_defs.h'

c   compute  dest = source(index_1,index_2,index_3,index_4)
c   gets source (4-dimensional) for dest (n-dimensional)
c   the dest and index fields must be in the same vp set
c
c parameters
c    dest1      : real destination field; n-dimensional
c    dest2      : real destination field; n-dimensional
c    dest3      : real destination field; n-dimensional
c    dest4      : real destination field; n-dimensional
c    index_1    : integer field ; first index of source array
c    index_2    : integer field ; second index of source array
c    index_3    : integer field ; third index of source array
c    index_4    : integer field ; fourth index of source array
c    source1    : real source field; 3-dimensional
```

```
c     source2     : real source field; 3-dimensional
c     source3     : real source field; 3-dimensional
c     source4     : real source field; 3-dimensional

      integer dest1, dest2, dest3, dest4
      integer source1, source2, source3, source4
      integer dest1_id, dest2_id, dest3_id, dest4_id
      integer dest_vp_set,dest_geo
      integer index_1,index_1_id
      integer index_2,index_2_id
      integer index_3,index_3_id
      integer index_4,index_4_id
      integer source1_id, source2_id, source3_id, source4_id
      integer source_vp_set,source_geo
      integer get_field,length,temp,i1,i2,i3,i4
      integer i,rank

      integer source_temp_id, dest_temp_id

      dest1_id = cmf_get_field_id(dest1)
      dest2_id = cmf_get_field_id(dest2)
      dest3_id = cmf_get_field_id(dest3)
      dest4_id = cmf_get_field_id(dest4)

      index_1_id = cmf_get_field_id(index_1)
      index_2_id = cmf_get_field_id(index_2)
      index_3_id = cmf_get_field_id(index_3)
      index_4_id = cmf_get_field_id(index_4)
      source1_id = cmf_get_field_id(source1)
      source2_id = cmf_get_field_id(source2)
      source3_id = cmf_get_field_id(source3)
      source4_id = cmf_get_field_id(source4)

      if (dest1_id .eq. 0) then
         print *,
     +'Error, the dest1 argument to gather4_4d  is not on the CM'
         stop
      endif

      if (dest2_id .eq. 0) then
         print *,
     +'Error, the dest2 argument to gather4_4d  is not on the CM'
         stop
      endif

      if (dest3_id .eq. 0) then
         print *,
     +'Error, the dest3 argument to gather4_4d  is not on the CM'
```

```
      stop
 endif

 if (dest4_id .eq. 0) then
      print *,
+'Error, the dest4 argument to gather4_4d  is not on the CM'
      stop
 endif


 if (index_1_id .eq. 0) then
      print *,
+'Error, the index_1 argument to gather4_4d is not on the CM'
      stop
 endif

 if (index_2_id .eq. 0) then
      print *,
+'Error, the index_2 argument to gather4_4d is not on the CM'
      stop
 endif

 if (index_3_id .eq. 0) then
      print *,
+'Error, the index_3 argument to gather4_4d is not on the CM'
      stop
 endif

 if (index_4_id .eq. 0) then
      print *,
+'Error, the index_4 argument to gather4_4d is not on the CM'
      stop
 endif

 if (source1_id .eq. 0) then
      print *,
+'Error, the source1 argument to gather4_4d is not on the CM'
      stop
 endif

 if (source2_id .eq. 0) then
      print *,
+'Error, the source2 argument to gather4_4d is not on the CM'
      stop
 endif

 if (source3_id .eq. 0) then
      print *,
```

```
     +'Error, the source3 argument to gather4_4d is not on the CM'
        stop
      endif

      if (source4_id .eq. 0) then
        print *,
     +'Error, the source4 argument to gather4_4d is not on the CM'
        stop
      endif

      dest_vp_set = cmf_get_vp_set_id(dest1)
      dest_geo = cm_vp_set_geometry(dest_vp_set)

      source_vp_set = cmf_get_vp_set_id(source1)
      source_geo = cm_vp_set_geometry(source_vp_set)
      call cm_set_vp_set(source_vp_set)
      source_temp_id = CM_allocate_stack_field(4*32)
      call CM_u_move_1L(source_temp_id, source1_id, 32)
      call CM_u_move_1L(source_temp_id + 32, source2_id, 32)
      call CM_u_move_1L(source_temp_id + 64, source3_id, 32)
      call CM_u_move_1L(source_temp_id + 96, source4_id, 32)

      call cm_set_vp_set(dest_vp_set)

      get_field = cm_allocate_stack_field(32)
      temp = cm_allocate_stack_field(32)
      i1 = cm_allocate_stack_field(32)
      i2 = cm_allocate_stack_field(32)
      i3 = cm_allocate_stack_field(32)
      i4 = cm_allocate_stack_field(32)
      call cm_set_context()
      dest_temp_id = CM_allocate_stack_field(4*32)

c      select context for destination

      call cm_my_news_coordinate_1l(temp,0,32)
      call cm_u_eq_constant_1l(temp,0,32)
      call cm_logand_context_with_test()

      rank = cm_geometry_rank(dest_geo)
      do i=1,rank-1
        length = cmf_get_axis_extent(dest1,i-1)
        call CM_my_news_coordinate_1L(temp,i,32)
        call CM_u_lt_constant_1L(temp,length,32)
        call CM_logand_context_with_test()
      enddo

      call cm_u_move_zero_always_1l(get_field,32)
```

```
      call cm_u_subtract_constant_3_1l(i1,index_1_id,1,32)
      call cm_u_subtract_constant_3_1l(i2,index_2_id,1,32)
      call cm_u_subtract_constant_3_1l(i3,index_3_id,1,32)
      call cm_u_subtract_constant_3_1l(i4,index_4_id,1,32)
      call cm_deposit_news_coordinate_1l(source_geo, get_field,1,
     +                                   i1,32)
      call cm_deposit_news_coordinate_1l(source_geo, get_field,2,
     +                                   i2,32)
      call cm_deposit_news_coordinate_1l(source_geo, get_field,3,
     +                                   i3,32)
      call cm_deposit_news_coordinate_1l(source_geo, get_field,4,
     +                                   i4,32)
      call cm_get_1l(dest_temp_id,get_field,source_temp_id,4*32)
      call CM_u_move_1L(dest1_id, dest_temp_id, 32)
      call CM_u_move_1L(dest2_id, dest_temp_id + 32, 32)
      call CM_u_move_1L(dest3_id, dest_temp_id + 64, 32)
      call CM_u_move_1L(dest4_id, dest_temp_id + 96, 32)

      call cm_deallocate_stack_through(dest_temp_id)
      call CMF_set_is_modified(dest1,MODIF)
      call CMF_set_is_modified(dest2,MODIF)
      call CMF_set_is_modified(dest3,MODIF)
      call CMF_set_is_modified(dest4,MODIF)
      return
      end




      subroutine scatter_add_1(dest,index_1,source)
      include '/usr/include/cm/paris-configuration-fort.h'
      include '/usr/include/cm/CMF_defs.h'

c   compute  dest(index_1) = dest(index_1) + source
c   sends source (n-dimensional) to dest (1-dimensional)
c
c parameters
c    dest      : real destination field; 1-dimensional
c    index_1   : integer field ; first index of dest array
c    source    : real source field

      integer dest,index_1,source

      integer dest_id,index_1_id,source_id
      integer source_vp_set,dest_vp_set,dest_geo,source_geo,send_field
      integer length,temp,i1,rank,i
      integer type,slen,elen

      dest_id = cmf_get_field_id(dest)
```

```
index_1_id = cmf_get_field_id(index_1)
source_id = cmf_get_field_id(source)

if (dest_id .eq. 0) then
   print *,
+'Error, the dest argument to scatter_add_1  is not on the CM'
   stop
endif

if (index_1_id .eq. 0) then
   print *,
+'Error, the index_1 argument to scatter_add_1 is not on the CM'
   stop
endif

if (source_id .eq. 0) then
   print *,
+'Error, the source argument to scatter_add_1 is not on the CM'
   stop
endif

dest_vp_set = cmf_get_vp_set_id(dest)
dest_geo = cm_vp_set_geometry(dest_vp_set)

source_vp_set = cmf_get_vp_set_id(source)
source_geo = cm_vp_set_geometry(source_vp_set)
call cm_set_vp_set(source_vp_set)

call cm_set_context()

send_field = cm_allocate_stack_field(32)
temp = cm_allocate_stack_field(32)
i1 = cm_allocate_stack_field(32)

call cm_my_news_coordinate_1l(temp,0,32)
call cm_u_eq_constant_1l(temp,0,32)
call cm_logand_context_with_test()

rank = CM_geometry_rank(source_geo)
do i=1,rank-1
   length = cmf_get_axis_extent(source,i-1)
   call CM_my_news_coordinate_1L(temp,i,32)
   call CM_u_lt_constant_1L(temp,length,32)
   call CM_logand_context_with_test()
enddo

call cm_u_move_zero_always_1l(send_field,32)
call cm_s_subtract_constant_3_1l(i1,index_1_id,1,32)
```

```
      call cm_deposit_news_coordinate_1l(dest_geo,send_field,1,
     +                                   i1,32)

      type = cmf_get_data_type(dest)
      elen = cmf_get_exponent_len(dest)
      slen = cmf_get_integer_len(dest)

       if (type.eq.cmssl_float) then
          call cm_send_with_f_add_1l(dest_id,send_field,source_id,
     +                               23,8,cm_no_field)
       else if (type.eq.cmssl_s_integer) then
          call cm_send_with_s_add_1l(dest_id,send_field,source_id,
     +                               slen,cm_no_field)
       else if (type.eq.cmssl_u_integer) then
          call cm_send_with_u_add_1l(dest_id,send_field,source_id,
     +                               slen,cm_no_field)
       else
          write (*,*) ' *** scatter_add_1: bad array data type'
       endif

      call cm_deallocate_stack_through(send_field)
      call CMF_set_is_modified(dest,MODIF)
      return
      end

      subroutine scatter_min_1(dest,index_1,source)
      include '/usr/include/cm/paris-configuration-fort.h'
      include '/usr/include/cm/CMF_defs.h'

c   compute  dest(index_1) = min(dest(index_1), source)
c   sends source (n-dimensional) to dest (1-dimensional)
c
c parameters
c    dest      : real destination field; 1-dimensional
c    index_1   : integer field ; first index of dest array
c    source    : real source field

      integer dest,index_1,source

      integer dest_id,index_1_id,source_id
      integer source_vp_set,dest_vp_set,dest_geo,source_geo,send_field
      integer length,temp,i1,rank,i
      integer type,slen,elen

      dest_id = cmf_get_field_id(dest)
      index_1_id = cmf_get_field_id(index_1)
      source_id = cmf_get_field_id(source)
```

```
 if (dest_id .eq. 0) then
    print *,
+'Error, the dest argument to scatter_min_1  is not on the CM'
    stop
 endif

 if (index_1_id .eq. 0) then
    print *,
+'Error, the index_1 argument to scatter_min_1 is not on the CM'
    stop
 endif

 if (source_id .eq. 0) then
    print *,
+'Error, the source argument to scatter_min_1 is not on the CM'
    stop
 endif

 dest_vp_set = cmf_get_vp_set_id(dest)
 dest_geo = cm_vp_set_geometry(dest_vp_set)

 source_vp_set = cmf_get_vp_set_id(source)
 source_geo = cm_vp_set_geometry(source_vp_set)
 call cm_set_vp_set(source_vp_set)

 call cm_set_context()

 send_field = cm_allocate_stack_field(32)
 temp = cm_allocate_stack_field(32)
 i1 = cm_allocate_stack_field(32)

 call cm_my_news_coordinate_1l(temp,0,32)
 call cm_u_eq_constant_1l(temp,0,32)
 call cm_logand_context_with_test()

 rank = CM_geometry_rank(source_geo)
 do i=1,rank-1
    length = cmf_get_axis_extent(source,i-1)
    call CM_my_news_coordinate_1L(temp,i,32)
    call CM_u_lt_constant_1L(temp,length,32)
    call CM_logand_context_with_test()
 enddo

 call cm_u_move_zero_always_1l(send_field,32)
 call cm_s_subtract_constant_3_1l(i1,index_1_id,1,32)
 call cm_deposit_news_coordinate_1l(dest_geo,send_field,1,
+                                   i1,32)
```

```
      type = cmf_get_data_type(dest)
      elen = cmf_get_exponent_len(dest)
      slen = cmf_get_integer_len(dest)

       if (type.eq.cmssl_float) then
          call cm_send_with_f_min_1l(dest_id,send_field,source_id,
     +                               23,8,cm_no_field)
       else if (type.eq.cmssl_s_integer) then
          call cm_send_with_s_min_1l(dest_id,send_field,source_id,
     +                               slen,cm_no_field)
       else if (type.eq.cmssl_u_integer) then
          call cm_send_with_u_min_1l(dest_id,send_field,source_id,
     +                               slen,cm_no_field)
       else
          write (*,*) ' *** scatter_min_1: bad array data type'
       endif
      call cm_deallocate_stack_through(send_field)
      call CMF_set_is_modified(dest,MODIF)
      return
      end


      subroutine scatter_max_1(dest,index_1,source)
      include '/usr/include/cm/paris-configuration-fort.h'
      include '/usr/include/cm/CMF_defs.h'

c   compute  dest(index_1) = max(dest(index_1), source)
c   sends source (n-dimensional) to dest (1-dimensional)
c
c parameters
c    dest      : real destination field; 1-dimensional
c    index_1   : integer field ; first index of dest array
c    source    : real source field

      integer dest,index_1,source

      integer dest_id,index_1_id,source_id
      integer source_vp_set,dest_vp_set,dest_geo,source_geo,send_field
      integer length,temp,i1,rank,i
      integer type,slen,elen

      dest_id = cmf_get_field_id(dest)
      index_1_id = cmf_get_field_id(index_1)
      source_id = cmf_get_field_id(source)

      if (dest_id .eq. 0) then
         print *,
     +'Error, the dest argument to scatter_max_1  is not on the CM'
         stop
```

```
 endif

 if (index_1_id .eq. 0) then
    print *,
+'Error, the index_1 argument to scatter_max_1 is not on the CM'
    stop
 endif

 if (source_id .eq. 0) then
    print *,
+'Error, the source argument to scatter_max_1 is not on the CM'
    stop
 endif

 dest_vp_set = cmf_get_vp_set_id(dest)
 dest_geo = cm_vp_set_geometry(dest_vp_set)

 source_vp_set = cmf_get_vp_set_id(source)
 source_geo = cm_vp_set_geometry(source_vp_set)
 call cm_set_vp_set(source_vp_set)

 call cm_set_context()

 send_field = cm_allocate_stack_field(32)
 temp = cm_allocate_stack_field(32)
 i1 = cm_allocate_stack_field(32)

 call cm_my_news_coordinate_1l(temp,0,32)
 call cm_u_eq_constant_1l(temp,0,32)
 call cm_logand_context_with_test()

 rank = CM_geometry_rank(source_geo)
 do i=1,rank-1
    length = cmf_get_axis_extent(source,i-1)
    call CM_my_news_coordinate_1L(temp,i,32)
    call CM_u_lt_constant_1L(temp,length,32)
    call CM_logand_context_with_test()
 enddo

 call cm_u_move_zero_always_1l(send_field,32)
 call cm_s_subtract_constant_3_1l(i1,index_1_id,1,32)
 call cm_deposit_news_coordinate_1l(dest_geo,send_field,1,
+                                   i1,32)

 type = cmf_get_data_type(dest)
 elen = cmf_get_exponent_len(dest)
 slen = cmf_get_integer_len(dest)
```

```
      if (type.eq.cmssl_float) then
         call cm_send_with_f_max_1l(dest_id,send_field,source_id,
     +                              23,8,cm_no_field)
      else if (type.eq.cmssl_s_integer) then
         call cm_send_with_s_max_1l(dest_id,send_field,source_id,
     +                              slen,cm_no_field)
      else if (type.eq.cmssl_u_integer) then
         call cm_send_with_u_max_1l(dest_id,send_field,source_id,
     +                              slen,cm_no_field)
      else
         write (*,*) ' *** scatter_max_1: bad array data type'
      endif

      call cm_deallocate_stack_through(send_field)
      call CMF_set_is_modified(dest,MODIF)
      return
      end



      subroutine scatter_add_2(dest,index_1,index_2,source)
      include '/usr/include/cm/paris-configuration-fort.h'
      include '/usr/include/cm/CMF_defs.h'

c   compute  dest(index_1,index_2) = dest(index_1,index_2) + source
c   sends source (n-dimensional) to dest (2-dimensional)
c
c parameters
c    dest      : real destination field;2-dimensional
c    index_1   : integer field ; first index of dest array
c    index_2   : integer field ; second index of dest array
c    source    : real source field

      integer dest,index_1,index_2,source

      integer dest_id,index_1_id,index_2_id,source_id
      integer source_vp_set,dest_vp_set,dest_geo,source_geo,send_field
      integer length,temp,i1,i2,rank,i
      integer type,slen,elen

      dest_id = cmf_get_field_id(dest)
      index_1_id = cmf_get_field_id(index_1)
      index_2_id = cmf_get_field_id(index_2)
      source_id = cmf_get_field_id(source)

      if (dest_id .eq. 0) then
         print *,
     +'Error, the dest argument to scatter_add_2  is not on the CM'
         stop
```

```
 endif

 if (index_1_id .eq. 0) then
    print *,
+'Error, the index_1 argument to scatter_add_2 is not on the CM'
    stop
 endif

 if (index_2_id .eq. 0) then
    print *,
+'Error, the index_2 argument to scatter_add_2 is not on the CM'
    stop
 endif

 if (source_id .eq. 0) then
    print *,
+'Error, the source argument to scatter_add_2 is not on the CM'
    stop
 endif

 dest_vp_set = cmf_get_vp_set_id(dest)
 dest_geo = cm_vp_set_geometry(dest_vp_set)

 source_vp_set = cmf_get_vp_set_id(source)
 source_geo = cm_vp_set_geometry(source_vp_set)
 call cm_set_vp_set(source_vp_set)

 call cm_set_context()

 send_field = cm_allocate_stack_field(32)
 temp = cm_allocate_stack_field(32)
 i1 = cm_allocate_stack_field(32)
 i2 = cm_allocate_stack_field(32)

 call cm_my_news_coordinate_1l(temp,0,32)
 call cm_u_eq_constant_1l(temp,0,32)
 call cm_logand_context_with_test()

 rank = CM_geometry_rank(source_geo)
 do i=1,rank-1
    length = cmf_get_axis_extent(source,i-1)
    call CM_my_news_coordinate_1L(temp,i,32)
    call CM_u_lt_constant_1L(temp,length,32)
    call CM_logand_context_with_test()
 enddo

 call cm_u_move_zero_always_1l(send_field,32)
 call cm_s_subtract_constant_3_1l(i1,index_1_id,1,32)
```

```
      call cm_s_subtract_constant_3_1l(i2,index_2_id,1,32)
      call cm_deposit_news_coordinate_1l(dest_geo,send_field,1,
     +                                   i1,32)
      call cm_deposit_news_coordinate_1l(dest_geo,send_field,2,
     +                                   i2,32)

      type = cmf_get_data_type(dest)
      elen = cmf_get_exponent_len(dest)
      slen = cmf_get_integer_len(dest)

       if (type.eq.cmssl_float) then
          call cm_send_with_f_add_1l(dest_id,send_field,source_id,
     +                               23,8,cm_no_field)
       else if (type.eq.cmssl_s_integer) then
          call cm_send_with_s_add_1l(dest_id,send_field,source_id,
     +                               slen,cm_no_field)
       else if (type.eq.cmssl_u_integer) then
          call cm_send_with_u_add_1l(dest_id,send_field,source_id,
     +                               slen,cm_no_field)
       else
          write (*,*) ' *** scatter_add_2: bad array data type'
       endif

      call cm_deallocate_stack_through(send_field)
      call CMF_set_is_modified(dest,MODIF)
      return
      end


      subroutine scatter_min_2(dest,index_1,index_2,source)
      include '/usr/include/cm/paris-configuration-fort.h'
      include '/usr/include/cm/CMF_defs.h'

c   compute  dest(index_1,index_2) = min(dest(index_1,index_2),source)
c   sends source (n-dimensional) to dest (2-dimensional)
c
c parameters
c    dest      : real destination field;2-dimensional
c    index_1   : integer field ; first index of dest array
c    index_2   : integer field ; second index of dest array
c    source    : real source field

      integer dest,index_1,index_2,source

      integer dest_id,index_1_id,index_2_id,source_id
      integer source_vp_set,dest_vp_set,dest_geo,source_geo,send_field
      integer length,temp,i1,i2,rank,i
      integer type,slen,elen
```

```
 dest_id = cmf_get_field_id(dest)
 index_1_id = cmf_get_field_id(index_1)
 index_2_id = cmf_get_field_id(index_2)
 source_id = cmf_get_field_id(source)

 if (dest_id .eq. 0) then
    print *,
+'Error, the dest argument to scatter_min_2  is not on the CM'
    stop
 endif

 if (index_1_id .eq. 0) then
    print *,
+'Error, the index_1 argument to scatter_min_2 is not on the CM'
    stop
 endif

 if (index_2_id .eq. 0) then
    print *,
+'Error, the index_2 argument to scatter_min_2 is not on the CM'
    stop
 endif

 if (source_id .eq. 0) then
    print *,
+'Error, the source argument to scatter_min_2 is not on the CM'
    stop
 endif

 dest_vp_set = cmf_get_vp_set_id(dest)
 dest_geo = cm_vp_set_geometry(dest_vp_set)

 source_vp_set = cmf_get_vp_set_id(source)
 source_geo = cm_vp_set_geometry(source_vp_set)
 call cm_set_vp_set(source_vp_set)

 call cm_set_context()

 send_field = cm_allocate_stack_field(32)
 temp = cm_allocate_stack_field(32)
 i1 = cm_allocate_stack_field(32)
 i2 = cm_allocate_stack_field(32)

 call cm_my_news_coordinate_1l(temp,0,32)
 call cm_u_eq_constant_1l(temp,0,32)
 call cm_logand_context_with_test()

 rank = CM_geometry_rank(source_geo)
```

```
      do i=1,rank-1
         length = cmf_get_axis_extent(source,i-1)
         call CM_my_news_coordinate_1L(temp,i,32)
         call CM_u_lt_constant_1L(temp,length,32)
         call CM_logand_context_with_test()
      enddo

      call cm_u_move_zero_always_1l(send_field,32)
      call cm_s_subtract_constant_3_1l(i1,index_1_id,1,32)
      call cm_s_subtract_constant_3_1l(i2,index_2_id,1,32)
      call cm_deposit_news_coordinate_1l(dest_geo,send_field,1,
     +                                   i1,32)
      call cm_deposit_news_coordinate_1l(dest_geo,send_field,2,
     +                                   i2,32)
      type = cmf_get_data_type(dest)
      elen = cmf_get_exponent_len(dest)
      slen = cmf_get_integer_len(dest)

       if (type.eq.cmssl_float) then
          call cm_send_with_f_min_1l(dest_id,send_field,source_id,
     +                               23,8,cm_no_field)
       else if (type.eq.cmssl_s_integer) then
          call cm_send_with_s_min_1l(dest_id,send_field,source_id,
     +                               slen,cm_no_field)
       else if (type.eq.cmssl_u_integer) then
          call cm_send_with_u_min_1l(dest_id,send_field,source_id,
     +                               slen,cm_no_field)
       else
          write (*,*) ' *** scatter_min_2: bad array data type'
       endif

      call cm_deallocate_stack_through(send_field)
      call CMF_set_is_modified(dest,MODIF)
      return
      end


      subroutine scatter_max_2(dest,index_1,index_2,source)
      include '/usr/include/cm/paris-configuration-fort.h'
      include '/usr/include/cm/CMF_defs.h'

c   compute  dest(index_1,index_2) = max(dest(index_1,index_2),source)
c   sends source (n-dimensional) to dest (2-dimensional)
c
c parameters
c    dest      : real destination field;2-dimensional
c    index_1   : integer field ; first index of dest array
c    index_2   : integer field ; second index of dest array
c    source    : real source field
```

```
      integer dest,index_1,index_2,source

      integer dest_id,index_1_id,index_2_id,source_id
      integer source_vp_set,dest_vp_set,dest_geo,source_geo,send_field
      integer length,temp,i1,i2,rank,i
      integer type,slen,elen

      dest_id = cmf_get_field_id(dest)
      index_1_id = cmf_get_field_id(index_1)
      index_2_id = cmf_get_field_id(index_2)
      source_id = cmf_get_field_id(source)

      if (dest_id .eq. 0) then
         print *,
     +'Error, the dest argument to scatter_max_2  is not on the CM'
         stop
      endif

      if (index_1_id .eq. 0) then
         print *,
     +'Error, the index_1 argument to scatter_max_2 is not on the CM'
         stop
      endif

      if (index_2_id .eq. 0) then
         print *,
     +'Error, the index_2 argument to scatter_max_2 is not on the CM'
         stop
      endif

      if (source_id .eq. 0) then
         print *,
     +'Error, the source argument to scatter_max_2 is not on the CM'
         stop
      endif

      dest_vp_set = cmf_get_vp_set_id(dest)
      dest_geo = cm_vp_set_geometry(dest_vp_set)

      source_vp_set = cmf_get_vp_set_id(source)
      source_geo = cm_vp_set_geometry(source_vp_set)
      call cm_set_vp_set(source_vp_set)

      call cm_set_context()

      send_field = cm_allocate_stack_field(32)
      temp = cm_allocate_stack_field(32)
```

```
      i1 = cm_allocate_stack_field(32)
      i2 = cm_allocate_stack_field(32)

      call cm_my_news_coordinate_1l(temp,0,32)
      call cm_u_eq_constant_1l(temp,0,32)
      call cm_logand_context_with_test()

      rank = CM_geometry_rank(source_geo)
      do i=1,rank-1
         length = cmf_get_axis_extent(source,i-1)
         call CM_my_news_coordinate_1L(temp,i,32)
         call CM_u_lt_constant_1L(temp,length,32)
         call CM_logand_context_with_test()
      enddo

      call cm_u_move_zero_always_1l(send_field,32)
      call cm_s_subtract_constant_3_1l(i1,index_1_id,1,32)
      call cm_s_subtract_constant_3_1l(i2,index_2_id,1,32)
      call cm_deposit_news_coordinate_1l(dest_geo,send_field,1,
     +                                   i1,32)
      call cm_deposit_news_coordinate_1l(dest_geo,send_field,2,
     +                                   i2,32)
      type = cmf_get_data_type(dest)
      elen = cmf_get_exponent_len(dest)
      slen = cmf_get_integer_len(dest)

       if (type.eq.cmssl_float) then
          call cm_send_with_f_max_1l(dest_id,send_field,source_id,
     +                               23,8,cm_no_field)
       else if (type.eq.cmssl_s_integer) then
          call cm_send_with_s_max_1l(dest_id,send_field,source_id,
     +                               slen,cm_no_field)
       else if (type.eq.cmssl_u_integer) then
          call cm_send_with_u_max_1l(dest_id,send_field,source_id,
     +                               slen,cm_no_field)
       else
          write (*,*) ' *** scatter_max_2: bad array data type'
       endif
      call cm_deallocate_stack_through(send_field)
      call CMF_set_is_modified(dest,MODIF)
      return
      end


      subroutine scatter_add_3(dest,index_1,index_2,index_3,source)
      include '/usr/include/cm/paris-configuration-fort.h'
      include '/usr/include/cm/CMF_defs.h'

c     compute  dest(index_1,index_2,index_3) =
```

```
c                  dest(index_1,index_2,index_3) + source
c   sends source (n-dimensional) to dest (3-dimensional)
c
c parameters
c    dest      : real destination field;3-dimensional
c    index_1   : integer field ; first index of dest array
c    index_2   : integer field ; second index of dest array
c    index_3   : integer field ; third index of dest array
c    source    : real source field

      integer dest,index_1,index_2,index_3,source

      integer dest_id,index_1_id,index_2_id,index_3_id,source_id
      integer source_vp_set,dest_vp_set,dest_geo,source_geo,send_field
      integer length,temp,i1,i2,i3,rank,i
      integer type,slen,elen

      dest_id = cmf_get_field_id(dest)
      index_1_id = cmf_get_field_id(index_1)
      index_2_id = cmf_get_field_id(index_2)
      index_3_id = cmf_get_field_id(index_3)
      source_id = cmf_get_field_id(source)

      if (dest_id .eq. 0) then
         print *,
     +'Error, the dest argument to scatter_add_3  is not on the CM'
         stop
      endif

      if (index_1_id .eq. 0) then
         print *,
     +'Error, the index_1 argument to scatter_add_3 is not on the CM'
         stop
      endif

      if (index_2_id .eq. 0) then
         print *,
     +'Error, the index_2 argument to scatter_add_3 is not on the CM'
         stop
      endif

      if (index_3_id .eq. 0) then
         print *,
     +'Error, the index_3 argument to scatter_add_3 is not on the CM'
         stop
      endif

      if (source_id .eq. 0) then
```

```
      print *,
     +'Error, the source argument to scatter_add_3 is not on the CM'
      stop
     endif

     dest_vp_set = cmf_get_vp_set_id(dest)
     dest_geo = cm_vp_set_geometry(dest_vp_set)

     source_vp_set = cmf_get_vp_set_id(source)
     source_geo = cm_vp_set_geometry(source_vp_set)
     call cm_set_vp_set(source_vp_set)

     call cm_set_context()

     send_field = cm_allocate_stack_field(32)
     temp = cm_allocate_stack_field(32)
     i1 = cm_allocate_stack_field(32)
     i2 = cm_allocate_stack_field(32)
     i3 = cm_allocate_stack_field(32)

     call cm_my_news_coordinate_1l(temp,0,32)
     call cm_u_eq_constant_1l(temp,0,32)
     call cm_logand_context_with_test()

     rank = CM_geometry_rank(source_geo)
     do i=1,rank-1
        length = cmf_get_axis_extent(source,i-1)
        call CM_my_news_coordinate_1L(temp,i,32)
        call CM_u_lt_constant_1L(temp,length,32)
        call CM_logand_context_with_test()
     enddo

     call cm_u_move_zero_always_1l(send_field,32)
     call cm_s_subtract_constant_3_1l(i1,index_1_id,1,32)
     call cm_s_subtract_constant_3_1l(i2,index_2_id,1,32)
     call cm_s_subtract_constant_3_1l(i3,index_3_id,1,32)
     call cm_deposit_news_coordinate_1l(dest_geo,send_field,1,
    +                                   i1,32)
     call cm_deposit_news_coordinate_1l(dest_geo,send_field,2,
    +                                   i2,32)
     call cm_deposit_news_coordinate_1l(dest_geo,send_field,3,
    +                                   i3,32)

     type = cmf_get_data_type(dest)
     elen = cmf_get_exponent_len(dest)
     slen = cmf_get_integer_len(dest)

      if (type.eq.cmssl_float) then
```

```
          call cm_send_with_f_add_1l(dest_id,send_field,source_id,
     +                               23,8,cm_no_field)
          else if (type.eq.cmssl_s_integer) then
             call cm_send_with_s_add_1l(dest_id,send_field,source_id,
     +                               slen,cm_no_field)
          else if (type.eq.cmssl_u_integer) then
             call cm_send_with_u_add_1l(dest_id,send_field,source_id,
     +                               slen,cm_no_field)
          else
             write (*,*) ' *** scatter_add_3: bad array data type'
          endif
          call cm_deallocate_stack_through(send_field)
          call CMF_set_is_modified(dest,MODIF)
          return
          end


          subroutine scatter_min_3(dest,index_1,index_2,index_3,source)
          include '/usr/include/cm/paris-configuration-fort.h'
          include '/usr/include/cm/CMF_defs.h'

c   compute  dest(index_1,index_2,index_3) =
c                  min(dest(index_1,index_2,index_3),source)
c   sends source (n-dimensional) to dest (3-dimensional)
c
c parameters
c    dest      : real destination field;3-dimensional
c    index_1   : integer field ; first index of dest array
c    index_2   : integer field ; second index of dest array
c    index_3   : integer field ; third index of dest array
c    source    : real source field

          integer dest,index_1,index_2,index_3,source

          integer dest_id,index_1_id,index_2_id,index_3_id,source_id
          integer source_vp_set,dest_vp_set,dest_geo,source_geo,send_field
          integer length,temp,i1,i2,i3,rank,i
          integer type,slen,elen

          dest_id = cmf_get_field_id(dest)
          index_1_id = cmf_get_field_id(index_1)
          index_2_id = cmf_get_field_id(index_2)
          index_3_id = cmf_get_field_id(index_3)
          source_id = cmf_get_field_id(source)

          if (dest_id .eq. 0) then
             print *,
     +'Error, the dest argument to scatter_min_3  is not on the CM'
             stop
```

```
 endif

 if (index_1_id .eq. 0) then
    print *,
+'Error, the index_1 argument to scatter_min_3 is not on the CM'
    stop
 endif

 if (index_2_id .eq. 0) then
    print *,
+'Error, the index_2 argument to scatter_min_3 is not on the CM'
    stop
 endif

 if (index_3_id .eq. 0) then
    print *,
+'Error, the index_3 argument to scatter_min_3 is not on the CM'
    stop
 endif

 if (source_id .eq. 0) then
    print *,
+'Error, the source argument to scatter_min_3 is not on the CM'
    stop
 endif

 dest_vp_set = cmf_get_vp_set_id(dest)
 dest_geo = cm_vp_set_geometry(dest_vp_set)

 source_vp_set = cmf_get_vp_set_id(source)
 source_geo = cm_vp_set_geometry(source_vp_set)
 call cm_set_vp_set(source_vp_set)

 call cm_set_context()

 send_field = cm_allocate_stack_field(32)
 temp = cm_allocate_stack_field(32)
 i1 = cm_allocate_stack_field(32)
 i2 = cm_allocate_stack_field(32)
 i3 = cm_allocate_stack_field(32)

 call cm_my_news_coordinate_1l(temp,0,32)
 call cm_u_eq_constant_1l(temp,0,32)
 call cm_logand_context_with_test()

 rank = CM_geometry_rank(source_geo)
 do i=1,rank-1
    length = cmf_get_axis_extent(source,i-1)
```

```
         call CM_my_news_coordinate_1L(temp,i,32)
         call CM_u_lt_constant_1L(temp,length,32)
         call CM_logand_context_with_test()
       enddo

       call cm_u_move_zero_always_1l(send_field,32)
       call cm_s_subtract_constant_3_1l(i1,index_1_id,1,32)
       call cm_s_subtract_constant_3_1l(i2,index_2_id,1,32)
       call cm_s_subtract_constant_3_1l(i3,index_3_id,1,32)
       call cm_deposit_news_coordinate_1l(dest_geo,send_field,1,
      +                                   i1,32)
       call cm_deposit_news_coordinate_1l(dest_geo,send_field,2,
      +                                   i2,32)
       call cm_deposit_news_coordinate_1l(dest_geo,send_field,3,
      +                                   i3,32)
       type = cmf_get_data_type(dest)
       elen = cmf_get_exponent_len(dest)
       slen = cmf_get_integer_len(dest)

        if (type.eq.cmssl_float) then
           call cm_send_with_f_min_1l(dest_id,send_field,source_id,
      +                               23,8,cm_no_field)
        else if (type.eq.cmssl_s_integer) then
           call cm_send_with_s_min_1l(dest_id,send_field,source_id,
      +                               slen,cm_no_field)
        else if (type.eq.cmssl_u_integer) then
           call cm_send_with_u_min_1l(dest_id,send_field,source_id,
      +                               slen,cm_no_field)
        else
           write (*,*) ' *** scatter_min_3: bad array data type'
        endif
       call cm_deallocate_stack_through(send_field)
       call CMF_set_is_modified(dest,MODIF)
       return
       end


       subroutine scatter_max_3(dest,index_1,index_2,index_3,source)
       include '/usr/include/cm/paris-configuration-fort.h'
       include '/usr/include/cm/CMF_defs.h'

c   compute  dest(index_1,index_2,index_3) =
c                  max(dest(index_1,index_2,index_3),source)
c   sends source (n-dimensional) to dest (3-dimensional)
c
c parameters
c    dest      : real destination field;3-dimensional
c    index_1   : integer field ; first index of dest array
c    index_2   : integer field ; second index of dest array
```

```
c     index_3    : integer field ; third index of dest array
c     source     : real source field

      integer dest,index_1,index_2,index_3,source

      integer dest_id,index_1_id,index_2_id,index_3_id,source_id
      integer source_vp_set,dest_vp_set,dest_geo,source_geo,send_field
      integer length,temp,i1,i2,i3,rank,i
      integer type,elen,slen

      dest_id = cmf_get_field_id(dest)
      index_1_id = cmf_get_field_id(index_1)
      index_2_id = cmf_get_field_id(index_2)
      index_3_id = cmf_get_field_id(index_3)
      source_id = cmf_get_field_id(source)

      if (dest_id .eq. 0) then
         print *,
     +'Error, the dest argument to scatter_max_3  is not on the CM'
         stop
      endif

      if (index_1_id .eq. 0) then
         print *,
     +'Error, the index_1 argument to scatter_max_3 is not on the CM'
         stop
      endif

      if (index_2_id .eq. 0) then
         print *,
     +'Error, the index_2 argument to scatter_max_3 is not on the CM'
         stop
      endif

      if (index_3_id .eq. 0) then
         print *,.
     +'Error, the index_3 argument to scatter_max_3 is not on the CM'
         stop
      endif

      if (source_id .eq. 0) then
         print *,
     +'Error, the source argument to scatter_max_3 is not on the CM'
         stop
      endif

      dest_vp_set = cmf_get_vp_set_id(dest)
      dest_geo = cm_vp_set_geometry(dest_vp_set)
```

```
source_vp_set = cmf_get_vp_set_id(source)
source_geo = cm_vp_set_geometry(source_vp_set)
call cm_set_vp_set(source_vp_set)

call cm_set_context()

send_field = cm_allocate_stack_field(32)
temp = cm_allocate_stack_field(32)
i1 = cm_allocate_stack_field(32)
i2 = cm_allocate_stack_field(32)
i3 = cm_allocate_stack_field(32)

call cm_my_news_coordinate_1l(temp,0,32)
call cm_u_eq_constant_1l(temp,0,32)
call cm_logand_context_with_test()

rank = CM_geometry_rank(source_geo)
do i=1,rank-1
   length = cmf_get_axis_extent(source,i-1)
   call CM_my_news_coordinate_1L(temp,i,32)
   call CM_u_lt_constant_1L(temp,length,32)
   call CM_logand_context_with_test()
enddo

call cm_u_move_zero_always_1l(send_field,32)
call cm_s_subtract_constant_3_1l(i1,index_1_id 1,32)
call cm_s_subtract_constant_3_1l(i2,index_2_id,1,32)
call cm_s_subtract_constant_3_1l(i3,index_3_id,1,32)
call cm_deposit_news_coordinate_1l(dest_geo,send_field,1,
+                                  i1,32)
call cm_deposit_news_coordinate_1l(dest_geo,send_field,2,
+                                  i2,32)
call cm_deposit_news_coordinate_1l(dest_geo,send_field,3,
+                                  i3,32)
type = cmf_get_data_type(dest)
elen = cmf_get_exponent_len(dest)
slen = cmf_get_integer_len(dest)

 if (type.eq.cmssl_float) then
    call cm_send_with_f_max_1l(dest_id,send_field,source_id,
+                             23,8,cm_no_field)
 else if (type.eq.cmssl_s_integer) then
    call cm_send_with_s_max_1l(dest_id,send_field,source_id,
+                             slen,cm_no_field)
 else if (type.eq.cmssl_u_integer) then
    call cm_send_with_u_max_1l(dest_id,send_field,source_id,
+                             slen,cm_no_field)
```

```
 else
    write (*,*) ' *** scatter_max_3: bad array data type'
 endif
call cm_deallocate_stack_through(send_field)
call CMF_set_is_modified(dest,MODIF)
return
end
```

```
      subroutine begin_fast_array(array)
      integer array
      call transpose32(array)
      return
      end

      subroutine end_fast_array(array)
      integer array
      call transpose32(array)
      return
      end

      subroutine transpose32(array)

      include '/usr/include/cm/paris-configuration-fort.h'
      include '/usr/include/cm/CMF_defs.h'

      integer array, array_id, array_vps, array_geo, array_type

      if (CMF_get_home(array) .eq. HOME_CM_ONLY) then
         print *, 'Error in transpose32, arg not on CM'
      endif

      array_type = CMF_get_data_type(array)
      array_id = CMF_get_field_id(array)
      array_vps = CMF_get_vp_set_id(array)
      array_geo = CM_vp_set_geometry(array_vps)

      if ((array_type .eq. CMF_LOGICAL) .or.
     +    (array_type .eq. CMF_COMPLEX) .or.
     +    (array_type .eq. CMF_CHARACTER) .or.
     +    ((array_type .eq. CMF_FLOAT) .and.
     +     (CMF_get_significand_len(array) .gt. 23))) then
        print *, 'Error in transpose32, arg not 32 bits long'
      endif

      if ((CM_geometry_axis_off_chip_bits(array_geo, 1) .ne. 0) .or.
     +    (CM_geometry_axis_on_chip_bits(array_geo, 1) .ne. 0)) then
        print *, 'Error in transpose32, first dimension is not serial'
      endif

      call CM_set_vp_set(array_vps)
      call CM_transpose32_1_1L(array_id, 32)
      call CMF_set_is_modified(array, MODIF)
```

```
      return
      end

      subroutine fast_array_access(dest, array, index)

      include '/usr/include/cm/paris-configuration-fort.h'
      include '/usr/include/cm/CMF_defs.h'

      integer array,array_id,array_vps,array_geo,array_type,array_rank
      integer dest, dest_id, dest_vps, dest_geo, dest_type, dest_rank
      integer index,index_id,index_vps,index_geo,index_type,index_rank
      integer array_id_alias
      integer temp_id
      integer i

      if (CMF_get_home(dest) .eq. HOME_CM_ONLY) then
         print *, 'Error in fast_array_access, arg not on CM'
      endif

      dest_type = CMF_get_data_type(dest)
      dest_id = CMF_get_field_id(dest)
      dest_vps = CMF_get_vp_set_id(dest)
      dest_geo = CM_vp_set_geometry(dest_vps)
      dest_rank = CM_geometry_rank(dest_geo)

      if ((dest_type .eq. CMF_LOGICAL) .or.
     +    (dest_type .eq. CMF_COMPLEX) .or.
     +    (dest_type .eq. CMF_CHARACTER) .or.
     +    ((dest_type .eq. CMF_FLOAT) .and.
     +     (CMF_get_significand_len(dest) .gt. 23))) then
        print *, 'Error in fast_array_access, arg not 32 bits long'
      endif

      if (CMF_get_home(array) .eq. HOME_CM_ONLY) then
         print *, 'Error in fast_array_access, arg not on CM'
      endif

      array_type = CMF_get_data_type(array)
      array_id = CMF_get_field_id(array)
      array_vps = CMF_get_vp_set_id(array)
      array_geo = CM_vp_set_geometry(array_vps)

      if (array_type .ne. dest_type) then
        print *,
     +     'Error in fast_array_access, array not same type as dest'
      endif

      if ((CM_geometry_axis_off_chip_bits(array_geo, 1) .ne. 0) .or.
```

```
+     (CM_geometry_axis_on_chip_bits(array_geo, 1) .ne. 0)) then
    print *,
+      'Error in fast_array_access, first dimension is not serial'
  endif

  if (CMF_get_home(index) .eq. HOME_CM_ONLY) then
     print *, 'Error in fast_array_access, arg not on CM'
  endif

  index_type = CMF_get_data_type(index)
  index_id = CMF_get_field_id(index)
  index_vps = CMF_get_vp_set_id(index)
  index_geo = CM_vp_set_geometry(index_vps)
  index_rank = CM_geometry_rank(index_geo)

  if (index_vps .ne. dest_vps) then
     print *, 'Error in fast_array_access, arrays dont conform'
  endif

  if ((index_type .ne. CMF_U_INTEGER) .and.
+     (index_type .ne. CMF_S_INTEGER)) then
    print *, 'Error in fast_array_access, index not integer'
  endif

  call CM_set_vp_set(index_vps)
  call CM_set_context()

  temp_id = CM_allocate_stack_field(32)

  call CM_my_news_coordinate_1L(temp_id, 0, 32)
  call CM_u_eq_constant_1L(temp_id, 0, 32)
  call CM_logand_context_with_test()

  do i=1,index_rank-1
    if ((CM_geometry_axis_off_chip_bits(array_geo, i+1) .ne.
+          CM_geometry_axis_off_chip_bits(index_geo, i)) .or.
+       (CM_geometry_axis_on_chip_bits(array_geo, i+1) .ne.
+          CM_geometry_axis_on_chip_bits(index_geo, i)) .or.
+       (CM_geometry_axis_off_chip_pos(array_geo, i+1) .ne.
+          CM_geometry_axis_off_chip_pos(index_geo, i)) .or.
+       (CM_geometry_axis_on_chip_pos(array_geo, i+1) .ne.
+          CM_geometry_axis_on_chip_pos(index_geo, i)) .or.
+       (CMF_get_axis_extent(array, i) .ne.
+          CMF_get_axis_extent(index, i-1)) .or.
+       (CMF_get_axis_extent(dest, i-1) .ne.
+          CMF_get_axis_extent(index, i-1))) then
      print *, 'Error in fast_array_access, args dont conform'
    endif
```

```
  call CM_my_news_coordinate_1L(temp_id, i, 32)
  call CM_u_lt_constant_1L(temp_id,
+                          CMF_get_axis_extent(dest,i-1),32)
  call CM_logand_context_with_test()
 enddo

 array_id_alias = CM_make_field_alias(array_id)

 call CM_u_subtract_constant_3_1L(temp_id, index_id, 1, 32)
 call CM_aref32_2L(dest_id, array_id_alias, temp_id, 32, 32,
+     CMF_get_axis_extent(array, 1))

 call CM_remove_field_alias(array_id_alias)
 call CM_deallocate_stack_through(temp_id)
 call CMF_set_is_modified(dest, MODIF)

 return
 end

 subroutine fast_array_update(array, source, index)

 include '/usr/include/cm/paris-configuration-fort.h'
 include '/usr/include/cm/CMF_defs.h'

 integer array,array_id,array_vps,array_geo,array_type,array_rank
 integer source, source_id, source_vps, source_geo
 integer source_type, source_rank
 integer index,index_id,index_vps,index_geo,index_type,index_rank
 integer array_id_alias
 integer temp_id
 integer i

 if (CMF_get_home(source) .eq. HOME_CM_ONLY) then
    print *, 'Error in fast_array_update, arg not on CM'
 endif

 source_type = CMF_get_data_type(source)
 source_id = CMF_get_field_id(source)
 source_vps = CMF_get_vp_set_id(source)
 source_geo = CM_vp_set_geometry(source_vps)
 source_rank = CM_geometry_rank(source_geo)

 if ((source_type .eq. CMF_LOGICAL) .or.
+    (source_type .eq. CMF_COMPLEX) .or.
+    (source_type .eq. CMF_CHARACTER) .or.
+    ((source_type .eq. CMF_FLOAT) .and.
+     (CMF_get_significand_len(source) .gt. 23))) then
    print *, 'Error in fast_array_update, arg not 32 bits long'
```

```
      endif

      if (CMF_get_home(array) .eq. HOME_CM_ONLY) then
         print *, 'Error in fast_array_update, arg not on CM'
      endif

      array_type = CMF_get_data_type(array)
      array_id = CMF_get_field_id(array)
      array_vps = CMF_get_vp_set_id(array)
      array_geo = CM_vp_set_geometry(array_vps)

      if (array_type .ne. source_type) then
        print *,
     +     'Error in fast_array_update, array not same type as source'
      endif

      if ((CM_geometry_axis_off_chip_bits(array_geo, 1) .ne. 0) .or.
     +    (CM_geometry_axis_on_chip_bits(array_geo, 1) .ne. 0)) then
        print *,
     +     'Error in fast_array_update, first dimension is not serial'
      endif

      if (CMF_get_home(index) .eq. HOME_CM_ONLY) then
         print *, 'Error in fast_array_update, arg not on CM'
      endif

      index_type = CMF_get_data_type(index)
      index_id = CMF_get_field_id(index)
      index_vps = CMF_get_vp_set_id(index)
      index_geo = CM_vp_set_geometry(index_vps)
      index_rank = CM_geometry_rank(index_geo)

      if (index_vps .ne. source_vps) then
         print *, 'Error in fast_array_update, arrays dont conform'
      endif

      if ((index_type .ne. CMF_U_INTEGER) .and.
     +    (index_type .ne. CMF_S_INTEGER)) then
        print *, 'Error in fast_array_update, index not integer'
      endif

      call CM_set_vp_set(index_vps)
      call CM_set_context()

      temp_id = CM_allocate_stack_field(32)

      call CM_my_news_coordinate_1L(temp_id, 0, 32)
      call CM_u_eq_constant_1L(temp_id, 0, 32)
```

```
 call CM_logand_context_with_test()

 do i=1,index_rank-1
   if ((CM_geometry_axis_off_chip_bits(array_geo, i+1) .ne.
+          CM_geometry_axis_off_chip_bits(index_geo, i)) .or.
+      (CM_geometry_axis_on_chip_bits(array_geo, i+1) .ne.
+          CM_geometry_axis_on_chip_bits(index_geo, i)) .or.
+      (CM_geometry_axis_off_chip_pos(array_geo, i+1) .ne.
+          CM_geometry_axis_off_chip_pos(index_geo, i)) .or.
+      (CM_geometry_axis_on_chip_pos(array_geo, i+1) .ne.
+          CM_geometry_axis_on_chip_pos(index_geo, i)) .or.
+      (CMF_get_axis_extent(array, i) .ne.
+          CMF_get_axis_extent(index, i-1)) .or.
+      (CMF_get_axis_extent(source, i-1) .ne.
+          CMF_get_axis_extent(index, i-1))) then
     print *, 'Error in fast_array_update, args dont conform'
  endif
  call CM_my_news_coordinate_1L(temp_id, i, 32)
  call CM_u_lt_constant_1L(temp_id,
+                          CMF_get_axis_extent(source,i-1),32)
  call CM_logand_context_with_test()
 enddo

 array_id_alias = CM_make_field_alias(array_id)

 call CM_u_subtract_constant_3_1L(temp_id, index_id, 1, 32)
 call CM_aset32_2L(source_id, array_id_alias, temp_id, 32, 32,
+     CMF_get_axis_extent(array, 1))

 call CM_remove_field_alias(array_id_alias)
 call CM_deallocate_stack_through(temp_id)
 call CMF_set_is_modified(array, MODIF)

 return
 end

 subroutine fast_array_access_2d(dest, array, inx1, inx2)

 include '/usr/include/cm/paris-configuration-fort.h'
 include '/usr/include/cm/CMF_defs.h'

 integer array,array_id,array_vps,array_geo,array_type,array_rank
 integer dest, dest_id, dest_vps, dest_geo, dest_type, dest_rank
 integer inx1,inx1_id,inx1_vps,inx1_geo,inx1_type,inx1_rank
 integer inx2,inx2_id,inx2_vps,inx2_geo,inx2_type,inx2_rank
 integer array_id_alias
 integer temp_id
 integer i
```

```
      integer len1, len2

      if (CMF_get_home(dest) .eq. HOME_CM_ONLY) then
         print *, 'Error in fast_array_access_2d, arg not on CM'
      endif

      dest_type = CMF_get_data_type(dest)
      dest_id = CMF_get_field_id(dest)
      dest_vps = CMF_get_vp_set_id(dest)
      dest_geo = CM_vp_set_geometry(dest_vps)
      dest_rank = CM_geometry_rank(dest_geo)

      if ((dest_type .eq. CMF_LOGICAL) .or.
     +    (dest_type .eq. CMF_COMPLEX) .or.
     +    (dest_type .eq. CMF_CHARACTER) .or.
     +    ((dest_type .eq. CMF_FLOAT) .and.
     +     (CMF_get_significand_len(dest) .gt. 23))) then
        print *, 'Error in fast_array_access_2d, arg not 32 bits long'
      endif

      if (CMF_get_home(array) .eq. HOME_CM_ONLY) then
         print *, 'Error in fast_array_access_2d, arg not on CM'
      endif

      array_type = CMF_get_data_type(array)
      array_id = CMF_get_field_id(array)
      array_vps = CMF_get_vp_set_id(array)
      array_geo = CM_vp_set_geometry(array_vps)

      if (array_type .ne. dest_type) then
        print *,
     +     'Error in fast_array_access_2d, array not same type as dest'
      endif

      if ((CM_geometry_axis_off_chip_bits(array_geo, 1) .ne. 0) .or.
     +    (CM_geometry_axis_on_chip_bits(array_geo, 1) .ne. 0)) then
        print *,
     +    'Error in fast_array_access_2d, first dimension is not serial'
      endif

      if (CMF_get_home(inx1) .eq. HOME_CM_ONLY) then
         print *, 'Error in fast_array_access_2d, arg not on CM'
      endif

      if (CMF_get_home(inx2) .eq. HOME_CM_ONLY) then
         print *, 'Error in fast_array_access_2d, arg not on CM'
      endif
```

```
      inx1_type = CMF_get_data_type(inx1)
      inx2_type = CMF_get_data_type(inx2)
      inx1_id = CMF_get_field_id(inx1)
      inx2_id = CMF_get_field_id(inx2)
      inx1_vps = CMF_get_vp_set_id(inx1)
      inx2_vps = CMF_get_vp_set_id(inx2)
      inx1_geo = CM_vp_set_geometry(inx1_vps)
      inx2_geo = CM_vp_set_geometry(inx2_vps)
      inx1_rank = CM_geometry_rank(inx1_geo)
      inx2_rank = CM_geometry_rank(inx2_geo)

      if (inx1_vps .ne. dest_vps) then
         print *, 'Error in fast_array_access_2d, arrays dont conform'
      endif

      if (inx2_vps .ne. dest_vps) then
         print *, 'Error in fast_array_access_2d, arrays dont conform'
      endif

      if ((inx1_type .ne. CMF_U_INTEGER) .and.
     +    (inx1_type .ne. CMF_S_INTEGER)) then
        print *, 'Error in fast_array_access_2d, inx1 not integer'
      endif

      if ((inx2_type .ne. CMF_U_INTEGER) .and.
     +    (inx2_type .ne. CMF_S_INTEGER)) then
        print *, 'Error in fast_array_access_2d, inx2 not integer'
      endif

      call CM_set_vp_set(inx1_vps)
      call CM_set_context()

      temp_id = CM_allocate_stack_field(32)

      call CM_my_news_coordinate_1L(temp_id, 0, 32)
      call CM_u_eq_constant_1L(temp_id, 0, 32)
      call CM_logand_context_with_test()

      do i=1,inx1_rank-1
        if ((CM_geometry_axis_off_chip_bits(array_geo, i+2) .ne.
     +          CM_geometry_axis_off_chip_bits(inx1_geo, i)) .or.
     +      (CM_geometry_axis_on_chip_bits(array_geo, i+2) .ne.
     +          CM_geometry_axis_on_chip_bits(inx1_geo, i)) .or.
     +      (CM_geometry_axis_off_chip_pos(array_geo, i+2) .ne.
     +          CM_geometry_axis_off_chip_pos(inx1_geo, i)) .or.
     +      (CM_geometry_axis_on_chip_pos(array_geo, i+2) .ne.
     +          CM_geometry_axis_on_chip_pos(inx1_geo, i)) .or.
     +      (CMF_get_axis_extent(array, i+1) .ne.
```

```
+          CMF_get_axis_extent(inx1, i-1)) .or.
+       (CMF_get_axis_extent(dest, i-1) .ne.
+          CMF_get_axis_extent(inx1, i-1))) then
     print *, 'Error in fast_array_access_2d, args dont conform'
  endif
  call CM_my_news_coordinate_1L(temp_id, i, 32)
  call CM_u_lt_constant_1L(temp_id,
+                           CMF_get_axis_extent(dest,i-1),32)
  call CM_logand_context_with_test()
 enddo

 array_id_alias = CM_make_field_alias(array_id)
 len1 = CM_geometry_axis_length(array_geo,1)
 len2 = CM_geometry_axis_length(array_geo,2)

 call CM_u_subtract_constant_3_1L(temp_id, inx2_id, 1, 32)
 call CM_u_multiply_constant_2_1L(temp_id, len1, 32)
 call CM_u_add_2_1L(temp_id, inx1_id, 32)
 call CM_u_subtract_constant_2_1L(temp_id, 1, 32)
 call CM_aref32_2L(dest_id, array_id_alias, temp_id, 32, 32,
+                  len1*len2)

 call CM_remove_field_alias(array_id_alias)
 call CM_deallocate_stack_through(temp_id)
 call CMF_set_is_modified(dest, MODIF)

 return
 end

 subroutine fast_array_update_2d(array, source, inx1, inx2)

 include '/usr/include/cm/paris-configuration-fort.h'
 include '/usr/include/cm/CMF_defs.h'

 integer array,array_id,array_vps,array_geo,array_type,array_rank
 integer source, source_id, source_vps, source_geo
 integer source_type, source_rank
 integer inx1,inx1_id,inx1_vps,inx1_geo,inx1_type,inx1_rank
 integer inx2,inx2_id,inx2_vps,inx2_geo,inx2_type,inx2_rank
 integer array_id_alias
 integer temp_id
 integer i
 integer len1, len2

 if (CMF_get_home(source) .eq. HOME_CM_ONLY) then
    print *, 'Error in fast_array_update_2d, arg not on CM'
 endif
```

```
      source_type = CMF_get_data_type(source)
      source_id = CMF_get_field_id(source)
      source_vps = CMF_get_vp_set_id(source)
      source_geo = CM_vp_set_geometry(source_vps)
      source_rank = CM_geometry_rank(source_geo)

      if ((source_type .eq. CMF_LOGICAL) .or.
     +    (source_type .eq. CMF_COMPLEX) .or.
     +    (source_type .eq. CMF_CHARACTER) .or.
     +    ((source_type .eq. CMF_FLOAT) .and.
     +     (CMF_get_significand_len(source) .gt. 23))) then
        print *, 'Error in fast_array_update_2d, arg not 32 bits long'
      endif

      if (CMF_get_home(array) .eq. HOME_CM_ONLY) then
         print *, 'Error in fast_array_update_2d, arg not on CM'
      endif

      array_type = CMF_get_data_type(array)
      array_id = CMF_get_field_id(array)
      array_vps = CMF_get_vp_set_id(array)
      array_geo = CM_vp_set_geometry(array_vps)

      if (array_type .ne. source_type) then
        print *,
     +    'Error in fast_array_update_2d, array not same type as source'
      endif

      if ((CM_geometry_axis_off_chip_bits(array_geo, 1) .ne. 0) .or.
     +    (CM_geometry_axis_on_chip_bits(array_geo, 1) .ne. 0)) then
        print *,
     +    'Error in fast_array_update_2d, first dimension is not serial'
      endif

      if (CMF_get_home(inx1) .eq. HOME_CM_ONLY) then
         print *, 'Error in fast_array_update_2d, arg not on CM'
      endif

      if (CMF_get_home(inx2) .eq. HOME_CM_ONLY) then
         print *, 'Error in fast_array_update_2d, arg not on CM'
      endif

      inx1_type = CMF_get_data_type(inx1)
      inx2_type = CMF_get_data_type(inx2)
      inx1_id = CMF_get_field_id(inx1)
      inx2_id = CMF_get_field_id(inx2)
      inx1_vps = CMF_get_vp_set_id(inx1)
      inx2_vps = CMF_get_vp_set_id(inx2)
```

```
 inx1_geo = CM_vp_set_geometry(inx1_vps)
 inx2_geo = CM_vp_set_geometry(inx2_vps)
 inx1_rank = CM_geometry_rank(inx1_geo)
 inx2_rank = CM_geometry_rank(inx2_geo)

 if (inx1_vps .ne. source_vps) then
    print *, 'Error in fast_array_update_2d, arrays dont conform'
 endif

 if (inx2_vps .ne. source_vps) then
    print *, 'Error in fast_array_update_2d, arrays dont conform'
 endif

 if ((inx1_type .ne. CMF_U_INTEGER) .and.
+    (inx1_type .ne. CMF_S_INTEGER)) then
   print *, 'Error in fast_array_update_2d, inx1 not integer'
 endif

 if ((inx2_type .ne. CMF_U_INTEGER) .and.
+    (inx2_type .ne. CMF_S_INTEGER)) then
   print *, 'Error in fast_array_update_2d, inx2 not integer'
 endif

 call CM_set_vp_set(inx1_vps)
 call CM_set_context()

 temp_id = CM_allocate_stack_field(32)

 call CM_my_news_coordinate_1L(temp_id, 0, 32)
 call CM_u_eq_constant_1L(temp_id, 0, 32)
 call CM_logand_context_with_test()

 do i=1,inx1_rank-1
   if ((CM_geometry_axis_off_chip_bits(array_geo, i+2) .ne.
+          CM_geometry_axis_off_chip_bits(inx1_geo, i)) .or.
+      (CM_geometry_axis_on_chip_bits(array_geo, i+2) .ne.
+          CM_geometry_axis_on_chip_bits(inx1_geo, i)) .or.
+      (CM_geometry_axis_off_chip_pos(array_geo, i+2) .ne.
+          CM_geometry_axis_off_chip_pos(inx1_geo, i)) .or.
+      (CM_geometry_axis_on_chip_pos(array_geo, i+2) .ne.
+          CM_geometry_axis_on_chip_pos(inx1_geo, i)) .or.
+      (CMF_get_axis_extent(array, i+1) .ne.
+          CMF_get_axis_extent(inx1, i-1)) .or.
+      (CMF_get_axis_extent(source, i-1) .ne.
+          CMF_get_axis_extent(inx1, i-1))) then
     print *, 'Error in fast_array_update_2d, args dont conform'
  endif
  call CM_my_news_coordinate_1L(temp_id, i, 32)
```

```
  call CM_u_lt_constant_1L(temp_id,
+                          CMF_get_axis_extent(source,i-1),32)
  call CM_logand_context_with_test()
 enddo

 array_id_alias = CM_make_field_alias(array_id)
 len1 = CM_geometry_axis_length(array_geo,1)
 len2 = CM_geometry_axis_length(array_geo,2)

 call CM_u_subtract_constant_3_1L(temp_id, inx2_id, 1, 32)
 call CM_u_multiply_constant_2_1L(temp_id, len1, 32)
 call CM_u_add_2_1L(temp_id, inx1_id, 32)
 call CM_u_subtract_constant_2_1L(temp_id, 1, 32)

 call CM_aset32_2L(source_id, array_id_alias, temp_id, 32, 32,
+                  len1*len2)

 call CM_remove_field_alias(array_id_alias)
 call CM_deallocate_stack_through(temp_id)
 call CMF_set_is_modified(array, MODIF)

 return
 end
```

B4 Table Lookup Routines

```
integer function make_integer_lookup(array, length)
integer array, length

include '/usr/include/cm/paris-configuration-fort.h'
include '/usr/include/cm/CMF_defs.h'

integer result

result = 0
call _MAKE_INT_LOOKUP(result, array, length, 3)
make_integer_lookup = result

end function make_integer_lookup



integer function  make_real_lookup(array, length)
integer array, length

include '/usr/include/cm/paris-configuration-fort.h'
include '/usr/include/cm/CMF_defs.h'

integer result

result = 0
call _MAKE_REAL_LOOKUP(result, array, length, 4)
make_real_lookup = result

end function make_real_lookup


integer function  make_lookup_cm(cm_source_array,
+                  cm_index, length, cm_mask)
integer cm_source_array, cm_index, length, cm_mask

include '/usr/include/cm/paris-configuration-fort.h'
include '/usr/include/cm/CMF_defs.h'

integer result, temp_index, save_context, rank_i
integer mask_id, source_vp_set
result = 0

source_vp_set = CMF_get_vp_set_id(cm_source_array)
call CM_set_vp_set(source_vp_set)
```

```
      if ((source_vp_set .ne.
     +  CMF_get_vp_set_id(cm_source_array)) .or.
     +  (source_vp_set .ne. CMF_get_vp_set_id(cm_index)) .or.
     +  (source_vp_set .ne. CMF_get_vp_set_id(cm_mask))) then
      print *, 'Arrays do not all belong to the same vp-set. '
      return
      endif

      temp_index = CM_allocate_stack_field(32)
      save_context = CM_allocate_stack_field(1)
      mask_id = CMF_GET_FIELD_ID(cm_mask)

      call CM_store_context(save_context)
      call CM_set_context

      call CM_load_context(mask_id)
      call CM_my_news_coordinate_1L(temp_index, 0, 32)
      call CM_u_le_constant_1l(temp_index, 0, 32)
      call CM_logand_context_with_test
      do rank_i=1, CM_geometry_rank(
     +    CM_vp_set_geometry(source_vp_set)) - 1
call CM_my_news_coordinate_1L(temp_index, rank_i, 32)
         call CM_u_le_constant_1l(temp_index,
     +   CMF_get_axis_extent(cm_mask, (rank_i - 1)), 32)
         call CM_logand_context_with_test
      enddo

      call CM_store_context(mask_id)

      if (CMF_GET_DATA_TYPE(cm_source_array) .eq.
     +        CMSSL_FLOAT) then
      call _MAKE_LOOKUP_CM(result, CMF_GET_FIELD_ID(
     + cm_source_array), CMF_GET_FIELD_ID(cm_index),length, 4,
     + mask_id)
      else if ((CMF_GET_DATA_TYPE(cm_source_array) .eq.
     +      CMSSL_U_INTEGER) .or.
     +      (CMF_GET_DATA_TYPE(cm_source_array) .eq.
     +      CMSSL_S_INTEGER)) then
      call _MAKE_LOOKUP_CM(result, CMF_GET_FIELD_ID(
     + cm_source_array), CMF_GET_FIELD_ID(cm_index), length, 3,
     + mask_id)
      else
      print *, 'IMPROPER SOURCE TYPE'
      end if

      call CM_load_context(save_context)
      call CM_deallocate_stack_through(temp_index)
```

```
      make_lookup_cm = result

      end function make_lookup_cm

      subroutine free_lookup(lookup_table)
      integer lookup_table

      include '/usr/include/cm/paris-configuration-fort.h'
      include '/usr/include/cm/CMF_defs.h'

      call _FREE_LOOKUP(lookup_table)

      return
      end

      subroutine lookup(dest_cm_array, lookup_table,
     +     index, cm_mask)
      integer lookup_table, dest_element_type
      integer index, dest_cm_array
      integer cm_mask

      include '/usr/include/cm/paris-configuration-fort.h'
      include '/usr/include/cm/CMF_defs.h'

      dest_element_type = CMF_get_data_type(dest_cm_array)

      call _LOOKUP(CMF_get_field_id(dest_cm_array),
     +    lookup_table, CMF_get_field_id(index),
     +    CMF_get_field_id(cm_mask), dest_element_type)
      call CMF_set_is_modified(dest_cm_array,MODIF)
      return
      end
```

```
#include <stdio.h>
#include <cm/paris.h>
#include <cm/CMSS_object.h>
struct lut {int allocated_p; int size; CM_field_id_t cm_field; int field_type;};

typedef struct lut lut_t;

void make_lookup();
void lookup();
void free_lookup();

#if defined(sparc)
# define MAKE_INT_LOOKUP make_int_lookup_
```

```
# define MAKE_REAL_LOOKUP make_real_lookup_
# define MAKE_LOOKUP_CM make_lookup_cm_
# define FREE_LOOKUP free_lookup_
# define LOOKUP lookup_
#endif

char *malloc();

void MAKE_INT_LOOKUP(lut_pointer, array, length, lut_type)
  int *array;
  int *length, *lut_type;
  lut_t **lut_pointer; {
  lut_t *result;
  int *array_temp;
  CM_field_id_t temp, save_context, index;
  CM_vp_set_id_t save_vp_set;
  int i;

  array_temp = array;

  result = (lut_t *) malloc(sizeof(lut_t));
  result->allocated_p = 1;
  result->size = 32 * (1 + (*length - 1) / 32);
  result->cm_field = CM_allocate_heap_field_vp_set(result->size, CM_physical_vp_set());
  result->field_type = *lut_type;

  save_vp_set = CM_current_vp_set;
  CM_set_vp_set(CM_physical_vp_set());
  temp = CM_allocate_stack_field(32);
  save_context = CM_allocate_stack_field(1);
  index = CM_allocate_stack_field(16);

  CM_store_context(save_context);
  CM_set_context();
  CM_s_move_zero_1L(index, 16);
  CM_my_send_address_1L(temp);
  CM_s_eq_zero_1L(temp, 5);

  CM_logand_context_with_test();

  for (i=0; i<*length; array_temp++) {
    if (*lut_type == 3)
     CM_s_move_constant_1L(temp, *array_temp, 32);
    CM_s_move_constant_1L(index, i, 16);
    CM_aset32_shared_2L(temp, result->cm_field, index, 32, 16, result->size);
    i = i + 1;};

  CM_load_context(save_context);
```

```
  CM_deallocate_stack_through(temp);
  CM_set_vp_set(save_vp_set);
  *lut_pointer = result;
}

void MAKE_REAL_LOOKUP(lut_pointer, array, length, lut_type)
  float *array;
  int *length, *lut_type;
  lut_t **lut_pointer; {
  lut_t *result;
  float *array_temp;
  CM_field_id_t temp, save_context, index;
  CM_vp_set_id_t save_vp_set;
  int i;

  array_temp = array;

  result = (lut_t *) malloc(sizeof(lut_t));
  result->allocated_p = 1;
  result->size = 32 * (1 + (*length - 1) / 32);
  result->cm_field = CM_allocate_heap_field_vp_set(result->size, CM_physical_vp_set());
  result->field_type = *lut_type;

  save_vp_set = CM_current_vp_set;
  CM_set_vp_set(CM_physical_vp_set());
  temp = CM_allocate_stack_field(32);
  save_context = CM_allocate_stack_field(1);
  index = CM_allocate_stack_field(16);

  CM_store_context(save_context);
  CM_set_context();
  CM_s_move_zero_1L(index, 16);
  CM_my_send_address_1L(temp);
  CM_s_eq_zero_1L(temp, 5);

  CM_logand_context_with_test();

  for (i=0; i<*length; array_temp++) {
    CM_f_move_constant_1L(temp, *array_temp, 23, 8);
    CM_s_move_constant_1L(index, i, 16);
    CM_aset32_shared_2L(temp, result->cm_field, index, 32, 16, result->size);
    i = i + 1;};

  CM_load_context(save_context);
  CM_deallocate_stack_through(temp);
  CM_set_vp_set(save_vp_set);
  *lut_pointer = result;
}
```

```
void MAKE_LOOKUP_CM(lut_pointer, array, cm_index, length, lut_type, mask)
  CM_field_id_t *array,*cm_index,*mask;
  int *length, *lut_type;
  lut_t **lut_pointer; {
  lut_t *result;
  CM_field_id_t save_context, index;
  CM_field_id_t temp_index, temp_news_coord;
  CM_vp_set_id_t save_vp_set;

  save_context = CM_allocate_stack_field(1);
  CM_store_context(save_context);
  save_vp_set = CM_current_vp_set;
  CM_set_vp_set(CM_field_vp_set(*array));

  result = (lut_t *) malloc(sizeof(lut_t));
  result->allocated_p = 1;
  result->size = 32 * (1 + (*length - 1) / 32);
  result->cm_field = CM_allocate_heap_field(result->size);
  result->field_type = *lut_type;

  temp_index = CM_allocate_stack_field(32);

  CM_load_context(*mask);
  CM_u_move_1L(temp_index,*cm_index,32);
  CM_u_subtract_constant_2_1L(temp_index,1,32);
  CM_aset32_shared_2L(*array, result->cm_field, temp_index, 32, 16, result->size);
  CM_spread_with_logior_1L(result->cm_field, result->cm_field, 0, 32);
  CM_set_vp_set(save_vp_set);
  CM_load_context(save_context);
  CM_deallocate_stack_through(save_context);
  *lut_pointer = result;
}


void FREE_LOOKUP(lookup_table)
  lut_t **lookup_table; {
  if ((**lookup_ta'le).allocated_p) {
    CM_deallocate_heap_field((**lookup_table).cm_field);
    (**lookup_table).allocated_p = 0;
  }
  else
    printf("free_lookup: table already deallocated!\n");
}

void LOOKUP(cm_field_id, lookup_table, cm_index_id, cm_mask_id, dest_element_type)
  lut_t **lookup_table;
```

```
  CM_field_id_t *cm_field_id, *cm_mask_id, *cm_index_id;
  int *dest_element_type; {
  CM_field_id_t save_context, temp_index_id;
  CM_vp_set_id_t save_vp_set;

  if ((**lookup_table).field_type != *dest_element_type)
       { if (*dest_element_type == 3)
         printf("Lookup table not allocated as an integer!! Instruction failed.\n");
         else
         { if (*dest_element_type == 4)
 printf("Lookup table not allocated as a real!! Instruction failed.\n");
  else
          printf("Destination array not an integer or real!! Instruction failed.\n");
        }}
  else {

  save_context = CM_allocate_stack_field(1);
  CM_store_context(save_context);
  save_vp_set = CM_current_vp_set;
  CM_set_vp_set(CM_field_vp_set(*cm_field_id));
  temp_index_id = CM_allocate_stack_field(16);
  CM_set_context();
  CM_load_context(*cm_mask_id);
  CM_s_subtract_constant_3_1L(temp_index_id, *cm_index_id, 1, 16);
  CM_invert_context();
  CM_s_move_zero_1L(temp_index_id, 16);
  CM_invert_context();
  if ((**lookup_table).allocated_p)
   CM_aref32_shared_2L(*cm_field_id, (**lookup_table).cm_field, temp_index_id,
32, 16, (**lookup_table).size);
  else
    printf("Lookup table has been deallocated! Instruction failed.\n");
  CM_set_vp_set(save_vp_set);
  CM_load_context(save_context);
  CM_deallocate_stack_through(save_context);
};
}
```

B5 Order Routine

```
subroutine order (dest,source,axis,mask)
integer dest
real source
integer axis
integer mask
integer temp,type,entry_vp_set

include '/usr/include/cm/paris-configuration-fort.h'
include '/usr/include/cm/CMF_defs.h'

entry_vp_set = cm_current_vp_set()
call cm_set_vp_set (cmf_get_vp_set_id(mask))
call cm_load_context (cmf_get_field_id(mask))
temp = cmf_get_field_id (source)
type = cmf_get_data_type (source)
if (type .eq. cmssl_s_integer) then
   call cm_s_rank_2L(cmf_get_field_id(dest),temp,axis,
!        32,32,cmf_upwards,cmf_none,0)
   end if
if (type .eq. cmssl_float) then
   call cm_f_rank_2L(cmf_get_field_id(dest),temp,axis,
!        32,23,8,cmf_upwards,cmf_none,0)
endif
call CMF_set_is_modified(dest,MODIF)
call cm_set_vp_set (entry_vp_set)
end
```

B6 Scan Routines

```
      subroutine sum_scan(result, source,dir,dim,sbit,mask)
      integer result,source,dim,mask,sbit
      logical dir
      include  '/usr/include/cm/paris-configuration-fort.h'
      include '/usr/include/cm/CMF_defs.h'
      integer scandir,length

      if (dir) then
         scandir = cm_upward
      else
         scandir = cm_downward
      endif

      call cm_set_vp_set (cmf_get_vp_set_id(source))
      call cm_load_context(cmf_get_field_id(mask))
      if (cmf_get_data_type(result) .eq. cmf_s_integer) then
         call cm_scan_with_s_add_1l (cmf_get_field_id(result),
     !        cmf_get_field_id(source),
     !        dim,
     !        32,
     !        scandir,
     !        cm_inclusive,
     !        cm_start_bit,
     !        cmf_get_field_id(sbit))

      endif

      if ((cmf_get_data_type(result) .eq. cmf_float)) then
         call cm_scan_with_f_add_1l (cmf_get_field_id(result),
     !        cmf_get_field_id(source),
     !        dim,
     !        cmf_get_significand_len(source),
     !        cmf_get_exponent_len(source),
     !        scandir,
     !        cm_inclusive,
     !        cm_start_bit,
     !        cmf_get_field_id(sbit))
      endif


      if ((cmf_get_data_type(result) .eq. cmf_complex)) then
         call cm_scan_with_c_add_1l (cmf_get_field_id(result),
     !        cmf_get_field_id(source),
     !        dim,
```

```
!        cmf_get_significand_len(source),
!        cmf_get_exponent_len(source),
!        scandir,
!        cm_inclusive,
!        cm_start_bit,
!        cmf_get_field_id(sbit))
 endif

 call cmf_set_is_modified(result,MODIF)
 return
 end


 subroutine product_scan(result, source,dir,dim,sbit,mask)
 integer result,source,dim,mask,sbit
 logical dir
 include  '/usr/include/cm/paris-configuration-fort.h'
 include '/usr/include/cm/CMF_defs.h'
 integer scandir,length

 if (dir) then
    scandir = cm_upward
 else
    scandir = cm_downward
 endif

 call cm_set_vp_set (cmf_get_vp_set_id(source))
 call cm_load_context(cmf_get_field_id(mask))

 if ((cmf_get_data_type(result) .eq. cmf_float)) then
    call cm_scan_with_f_multiply_1l (cmf_get_field_id(result),
!        cmf_get_field_id(source),
!        dim,
!        cmf_get_significand_len(source),
!        cmf_get_exponent_len(source),
!        scandir,
!        cm_inclusive,
!        cm_start_bit,
!        cmf_get_field_id(sbit))
 else
 print *, 'nrl-cmf-lib scans: integer products not supported'
 end if

 call cmf_set_is_modified(result,MODIF)
 return
 end
```

```
      subroutine max_scan(result, source,dir,dim,sbit,mask)
      integer result,source,dim,mask,sbit
      logical dir
      include  '/usr/include/cm/paris-configuration-fort.h'
      include '/usr/include/cm/CMF_defs.h'
      integer scandir,length

      if (dir) then
         scandir = cm_upward
      else
         scandir = cm_downward
      endif

      call cm_set_vp_set (cmf_get_vp_set_id(source))
      call cm_load_context(cmf_get_field_id(mask))
      if (cmf_get_data_type(result) .eq. cmf_s_integer) then
         call cm_scan_with_s_max_1l (cmf_get_field_id(result),
     !        cmf_get_field_id(source),
     !        dim,
     !        32,
     !        scandir,
     !        cm_inclusive,
     !        cm_start_bit,
     !        cmf_get_field_id(sbit))

      endif

      if ((cmf_get_data_type(result) .eq. cmf_float)) then
         call cm_scan_with_f_max_1l (cmf_get_field_id(result),
     !        cmf_get_field_id(source),
     !        dim,
     !        cmf_get_significand_len(source),
     !        cmf_get_exponent_len(source),
     !        scandir,
     !        cm_inclusive,
     !        cm_start_bit,
     !        cmf_get_field_id(sbit))
      endif

      call cmf_set_is_modified(result,MODIF)
      return
      end



      subroutine min_scan(result, source,dir,dim,sbit,mask)
```

```
      integer result,source,dim,mask,sbit
      logical dir
      include  '/usr/include/cm/paris-configuration-fort.h'
      include '/usr/include/cm/CMF_defs.h'
      integer scandir,length

      if (dir) then
         scandir = cm_upward
      else
         scandir = cm_downward
      endif

      call cm_set_vp_set (cmf_get_vp_set_id(source))
      call cm_load_context(cmf_get_field_id(mask))
      if (cmf_get_data_type(result) .eq. cmf_s_integer) then
         call cm_scan_with_s_min_1l (cmf_get_field_id(result),
     !        cmf_get_field_id(source),
     !        dim,
     !        32,
     !        scandir,
     !        cm_inclusive,
     !        cm_start_bit,
     !        cmf_get_field_id(sbit))

      endif

      if ((cmf_get_data_type(result) .eq. cmf_float)) then
         call cm_scan_with_f_min_1l (cmf_get_field_id(result),
     !        cmf_get_field_id(source),
     !        dim,
     !        cmf_get_significand_len(source),
     !        cmf_get_exponent_len(source),
     !        scandir,
     !        cm_inclusive,
     !        cm_start_bit,
     !        cmf_get_field_id(sbit))
      endif

      call cmf_set_is_modified(result,MODIF)
      return
      end



      subroutine or_scan(result, source,dir,dim,sbit,mask)
      integer result,source,dim,mask,sbit
      logical dir
      include  '/usr/include/cm/paris-configuration-fort.h'
```

```
include '/usr/include/cm/CMF_defs.h'
integer scandir,length


if (dir) then
   scandir = cm_upward
else
   scandir = cm_downward
endif

if (cmf_get_data_type(result) .eq. cmf_logical) then
   length=1
endif

if ((cmf_get_data_type(result) .eq. cmf_u_integer) .or.
!     (cmf_get_data_type(result) .eq. cmf_s_integer)) then
   length=32
endif

call cm_set_vp_set (cmf_get_vp_set_id(source))
call cm_load_context(cmf_get_field_id(mask))
call cm_scan_with_logior_1l (cmf_get_field_id(result),
!     cmf_get_field_id(source),
!     dim,
!     length,
!     scandir,
!     cm_inclusive,
!     cm_start_bit,
!     cmf_get_field_id(sbit))

call cmf_set_is_modified(result,MODIF)
return
end



subroutine xor_scan(result, source,dir,dim,sbit,mask)
integer result,source,dim,mask,sbit
logical dir
include  '/usr/include/cm/paris-configuration-fort.h'
include '/usr/include/cm/CMF_defs.h'
integer scandir,length


if (dir) then
   scandir = cm_upward
else
   scandir = cm_downward
```

```
 endif

 if (cmf_get_data_type(result) .eq. cmf_logical) then
    length=1
 endif
 if ((cmf_get_data_type(result) .eq. cmf_u_integer) .or.
!     (cmf_get_data_type(result) .eq. cmf_s_integer)) then
    length=32
 endif

 call cm_set_vp_set (cmf_get_vp_set_id(source))
 call cm_load_context(cmf_get_field_id(mask))
 call cm_scan_with_logxor_1l (cmf_get_field_id(result),
!     cmf_get_field_id(source),
!     dim,
!     length,
!     scandir,
!     cm_inclusive,
!     cm_start_bit,
!     cmf_get_field_id(sbit))

 call cmf_set_is_modified(result,MODIF)
 return
 end



 subroutine and_scan(result, source,dir,dim,sbit,mask)
 integer result,source,dim,mask,sbit
 logical dir
 include  '/usr/include/cm/paris-configuration-fort.h'
 include '/usr/include/cm/CMF_defs.h'
 integer scandir,length


 if (dir) then
    scandir = cm_upward
 else
    scandir = cm_downward
 endif


 if (cmf_get_data_type(result) .eq. cmf_logical) then
    length=1
 endif
 if ((cmf_get_data_type(result) .eq. cmf_u_integer) .or.
!     (cmf_get_data_type(result) .eq. cmf_s_integer)) then
    length=32
```

```
 endif

 call cm_set_vp_set (cmf_get_vp_set_id(source))
 call cm_load_context(cmf_get_field_id(mask))
 call cm_scan_with_logand_1l (cmf_get_field_id(result),
!     cmf_get_field_id(source),
!     dim,
!     length,
!     scandir,
!     cm_inclusive,
!     cm_start_bit,
!     cmf_get_field_id(sbit))

 call cmf_set_is_modified(result,MODIF)
 return
 end



 subroutine copy_scan(result, source,dir,dim,sbit,mask)
 integer result,source,dim,mask,sbit
 logical dir
 include  '/usr/include/cm/paris-conf' uration-fort.h'
 include '/usr/include/cm/CMF_defs.h'
 integer scandir,length

 if (dir) then
    scandir = cm_upward
 else
    scandir = cm_downward
 endif

 if (cmf_get_data_type(result) .eq. cmf_logical) then
    length=1
 endif

 if ((cmf_get_data_type(result) .eq. cmf_u_integer) .or.
!     (cmf_get_data_t' (result) .eq. cmf_s_integer)) then
    length=32
 endif


 if (cmf_get_data_type(result) .eq. cmf_float) then
     length =cmf_get_significand_len(source)+
!     cmf_get_exponent_len(source)+1
 endif
```

```
 if (cmf_get_data_type(result) .eq. cmf_complex) then
    length =2*(cmf_get_significand_len(source)+
!         cmf_get_exponent_len(source)+1)
 endif

 call cm_set_vp_set (cmf_get_vp_set_id(source))
 call cm_load_context(cmf_get_field_id(mask))
 call cm_scan_with_copy_1l (cmf_get_field_id(result),
!      cmf_get_field_id(source),
!      dim,
!      length,
!      scandir,
!      cm_inclusive,
!      cm_start_bit,
!      cmf_get_field_id(sbit))

 call cmf_set_is_modified(result,MODIF)
 return
 end
```

B7 Framebuffer Routines

```
      subroutine init_fb(x_dim, y_dim)

      include '/usr/include/cm/paris-configuration-fort.h'
      include '/usr/include/cm/CMF_defs.h'

      common /display_common/ my_geometry, my_vp_set, my_color,
     +     my_display, ntsc_on
      integer my_geometry, my_vp_set, my_color, my_display
      integer ntsc_on

      integer x_dim, y_dim
      integer dims(2)
CMF$  LAYOUT DIMS(:SERIAL)
      integer zoom
      logical kludge(256,256)
cmf$  layout kludge(:news,:news)
      integer physical_x, physical_y, foo
      character*10 a_null

      kludge =.true.
      dims(1) = x_dim
      dims(2) = y_dim
      my_geometry = CM_create_geometry(dims, 2)
      my_vp_set = CM_allocate_vp_set(my_geometry)

      call _attach_fb(my_display, ntsc_on)
      call CMFB_initialize_display(my_display, 8, 1)
      physical_x = CMFB_width(my_display)
      physical_y = CMFB_height(my_display)
      zoom = physical_x / x_dim
      if (zoom .gt. physical_y/y_dim) then
         zoom = physical_y / y_dim
      endif
      if (zoom .gt. 0) then
         zoom = zoom - 1
      endif
      call CMFB_set_zoom(my_display, zoom, zoom, 0)
      if (ntsc_on  .eq. 1) then
      call CMFB_set_pan(my_display, -32/(zoom+1),0)
      endif
      call CM_set_vp_set(my_vp_set)
      my_color = CM_allocate_heap_field(8)
      call CM_u_move_zero_always_1L(my_color, 8)
      return
      end
```

```
      subroutine plot_x_y(x, y, color, mask)

      integer x, y, color, mask

      include '/usr/include/cm/paris-configuration-fort.h'
      include '/usr/include/cm/CMF_defs.h'

      common /display_common/ my_geometry, my_vp_set, my_color,
     +     my_display, ntsc_on
      integer my_geometry, my_vp_set, my_color, my_display
      integer ntsc_on

      integer x_id, y_id, color_id, mask_id, rank_i
      integer the_buffer
      integer old_vp_set
      integer a_send_address
      integer temp_index

      x_id = CMF_get_field_id(x)
      y_id = CMF_get_field_id(y)
      color_id = CMF_get_field_id(color)
      mask_id = CMF_get_field_id(mask)
      old_vp_set = CMF_get_vp_set_id(x)

      if ((old_vp_set .ne. CMF_get_vp_set_id(y)) .or.
     +    (old_vp_set .ne. CMF_get_vp_set_id(color)) .or.
     +    (old_vp_set .ne. CMF_get_vp_set_id(mask))) then
      print *, 'Arrays do not all belong to the same vp-set. '
      return
      endif

      call CM_set_vp_set(my_vp_set)
      call CM_u_move_zero_always_1L(my_color, 8)
      call CM_set_vp_set(old_vp_set)
      call CM_set_context
      temp_index = CM_allocate_stack_field(32)
      call CM_load_context(mask_id)
      call CM_my_news_coordinate_1L(temp_index, 0, 32)
      call CM_u_le_constant_1l(temp_index, 0, 32)
      call CM_logand_context_with_test
      do rank_i=1, CM_geometry_rank(
     +    CM_vp_set_geometry(old_vp_set)) - 1
call CM_my_news_coordinate_1L(temp_index, rank_i, 32)
         call CM_u_le_constant_1l(temp_index,
     +        CMF_get_axis_extent(y, (rank_i - 1)), 32)
         call CM_logand_context_with_test
      enddo
```

```
 call CM_deallocate_stack_through(temp_index)

 a_send_address = CM_allocate_stack_field(32)
 call CMFB_shuffle_from_x_y(a_send_address, x_id, y_id,
+     my_geometry)
 call CM_send_1L(my_color, a_send_address, color_id, 8,
+     CM_no_field)
 call CM_set_vp_set(my_vp_set)
 the_buffer = CMFB_spare_buffer(my_display)
 call CMFB_write_preshuffled_always(my_display, the_buffer,
+ my_color, 0, 0)

 call CMFB_switch_buffer(my_display, the_buffer)
 call CM_deallocate_stack_through(a_send_address)
 call CM_set_vp_set(old_vp_set)
 return
 end


 subroutine plot_x_y_over(x, y, color, mask)

 integer x, y, color, mask

 include '/usr/include/cm/paris-configuration-fort.h'
 include '/usr/include/cm/CMF_defs.h'

 common /display_common/ my_geometry, my_vp_set, my_color,
+     my_display, ntsc_on
 integer my_geometry, my_vp_set, my_color, my_display
 integer ntsc_on

 integer x_id, y_id, color_id, mask_id, rank_i
 integer the_buffer
 integer old_vp_set
 integer a_send_address
 integer temp_index

 x_id = CMF_get_field_id(x)
 y_id = CMF_get_field_id(y)
 color_id = CMF_get_field_id(color)
 mask_id = CMF_get_field_id(mask)
 old_vp_set = CMF_get_vp_set_id(x)

 if ((old_vp_set .ne. CMF_get_vp_set_id(y)) .or.
+    (old_vp_set .ne. CMF_get_vp_set_id(color)) .or.
+    (old_vp_set .ne. CMF_get_vp_set_id(mask))) then
 print *, 'Arrays do not all belong to the same vp-set. '
 return
```

```
      endif

      call CM_set_vp_set(old_vp_set)
      call CM_set_context
      temp_index = CM_allocate_stack_field(32)
      call CM_load_context(mask_id)
      call CM_my_news_coordinate_1L(temp_index, 0, 32)
      call CM_u_le_constant_1l(temp_index, 0, 32)
      call CM_logand_context_with_test
      do rank_i=1, CM_geometry_rank(
     +    CM_vp_set_geometry(old_vp_set)) - 1
call CM_my_news_coordinate_1L(temp_index, rank_i, 32)
         call CM_u_le_constant_1l(temp_index,
     +       CMF_get_axis_extent(y, (rank_i - 1)), 32)
         call CM_logand_context_with_test
      enddo
      call CM_deallocate_stack_through(temp_index)

      a_send_address = CM_allocate_stack_field(32)
      call CMFB_shuffle_from_x_y(a_send_address, x_id, y_id,
     +     my_geometry)
      call CM_send_1L(my_color, a_send_address, color_id, 8,
     +     CM_no_field)
      call CM_set_vp_set(my_vp_set)
      the_buffer = CMFB_spare_buffer(my_display)
      call CMFB_write_preshuffled_always(my_display, the_buffer,
     + my_color, 0, 0)

      call CMFB_switch_buffer(my_display, the_buffer)
      call CM_deallocate_stack_through(a_send_address)
      call CM_set_vp_set(old_vp_set)
      return
      end


      subroutine release_frame_buffer()
      include '/usr/include/cm/paris-configuration-fort.h'
      include '/usr/include/cm/CMF_defs.h'

      common /display_common/ my_geometry, my_vp_set, my_color,
     +     my_display, ntsc_on
      integer my_geometry, my_vp_set, my_color, my_display
      integer ntsc_on

      call CMFB_detach_display(my_display)
      call CM_deallocate_heap_field(my_color)
      call CM_deallocate_vp_set(my_vp_set)
      call CM_deallocate_geometry(my_geometry)
```

```
      return
      end


      subroutine set_color(color_id, red, green, blue)
      integer color_id, red, green, blue

      include '/usr/include/cm/paris-configuration-fort.h'
      include '/usr/include/cm/CMF_defs.h'

      common /display_common/ my_geometry, my_vp_set, my_color,
     +     my_display, ntsc_on
      integer my_geometry, my_vp_set, my_color, my_display
      integer ntsc_on

      call CMFB_write_color(my_display, CMFB_red, color_id, red)
      call CMFB_write_color(my_display, CMFB_green, color_id, green)
      call CMFB_write_color(my_display, CMFB_blue, color_id, blue)
      return
      end


      subroutine plot_from_grid(color)
      integer color

      include '/usr/include/cm/paris-configuration-fort.h'
      include '/usr/include/cm/CMF_defs.h'

      common /display_common/ my_geom(  y, my_vp_set, my_color,
     +     my_display, ntsc_on
      integer my_geometry, my_vp_set, my_color, my_display
      integer ntsc_on

      integer geometry_old, geometry_new
      integer dimensions(2)
      integer color_id
      integer the_buffer
      integer color_vp_set_id

      color_id = CMF_get_field_id(color)
      color_vp_set_id = CMF_get_vp_set_id(color)

      dimensions(1) = CM_geometry_axis_length
     +     (CM_vp_set_geometry(color_vp_set_id), 1)
      dimensions(2) = CM_geometry_axis_length
     +     (CM_vp_set_geometry (color_vp_set_id), 2)

      geometry_old = CM_vp_set_geometry(color_vp_set_id)
```

```
      geometry_new = CM_intern_geometry(dimensions, 2)
      call CM_set_vp_set_geometry(color_vp_set_id,geometry_new)
      the_buffer = CMFB_spare_buffer(my_display)
      call CMFB_write_always(my_display, the_buffer, color_id, 0, 0)
      call CM_set_context ()
      call CMFB_switch_buffer(my_display, the_buffer)
      call CM_set_vp_set_geometry(color_vp_set_id, geometry_old)
      return
      end
```

```
#include <cm/paris.h>
#include <cm/cmfb.h>

#if defined(sparc)
# define ATTACH_FB attach_fb_
#endif

char *getenv();

void ATTACH_FB(display, ntsc_on)
CMFB_display_id_t *display;
int *ntsc_on;
{
  char * fb_type;
  *ntsc_on = 0;
  *display = CMFB_attach_display(getenv("CM_FRAMEBUFFER"), 0);
  fb_type = getenv("CM_FB_MODE");
  if (fb_type && !strcmp(fb_type, "NTSC")) {
    *ntsc_on = 1;
    CMFB_set_monitor_id(*display, CMFB_ntsc);
  }
}
```

B8 Plot Routines

```
c     this is a general purpose routine for using
c     paris from cmf. it sets vp set and the context
c     to reflect the processor configuration in array 'x'
c
      subroutine configure(x)
      integer x
      integer address,i
      include '/usr/include/cm/paris-configuration-fort.h'
      include '/usr/include/cm/CMF_defs.h'

      call cm_set_vp_set (cmf_get_vp_set_id(x))
      call cm_set_context()
      address = cm_allocate_stack_field(32)
      call cm_my_news_coordinate_1l (address,0,32)
      call cm_u_eq_constant_1l(address,0,32)
      call cm_logand_context_with_test()
      do 2 i=1,cmf_get_rank(1)
         call cm_my_news_coordinate_1l (address,i,32)
         call cm_u_lt_constant_1l(address,cmf_get_axis_extent(1,i-1),32)
         call cm_logand_context_with_test()
2     continue
      call cm_deallocate_stack_through (address)
      end

      subroutine openpl()
      common /scale_common/ sx0,sy0,sx1,sy1
      real sx0,sy0,sx1,sy1
      sx0=0.0
      sx1=1024.0
      sy0=0.0
      sy1=1024.0
      call init_fb(1024,1024)
      call _attach(1024,1024)
      end

      subroutine closepl()
      call release_frame_buffer()
      call _detach()
      end

      subroutine erase()
      common /display_common/ my_geometry,my_vp_set,my_color,my_display
      integer my_geometry, my_vp_set, my_color, my_display
      call _clear(my_color)
      end
```

```
      subroutine set_text_size(x)
      integer x
      call _set_font_number(x)
      end

      subroutine space(x1,y1,x2,y2)
      real x1,y1,x2,y2
      common /scale_common/ sx0,sy0,sx1,sy1
      real sx0,sy0,sx1,sy1
      sx0=x1
      sy0=y1
      sx1=x2
      sy1=y2
      end

      subroutine scale(x,y)
      include '/usr/include/cm/paris-configuration-fort.h'
      include '/usr/include/cm/CMF_defs.h'
      integer x,y
      real plot_x,ploy_y
      common /scale_common/ sx0,sy0,sx1,sy1
      real sx0,sy0,sx1,sy1
      common /display_common/ my_geometry,my_vp_set,my_color,my_display
      integer my_geometry, my_vp_set, my_color, my_display

      plot_x = real(cm_geometry_axis_length(my_geometry,0))
      plot_y = real(cm_geometry_axis_length(my_geometry,1))
      if ((sx0 .eq. 0.0) .and. (sx1 .eq. 0.0) .and. (sy1 .eq. 0.0)
     !     .and. (sy0 .eq. 0.0)) then
          sx0=0.0
          sy0=0.0
          sx1=plot_x
          sy1=plot_y
          endif
      call CM_f_subtract_constant_2_1L (x,dble(sx0),23,8)
      call CM_f_subtract_constant_2_1L (y,dble(sy0),23,8)
      call CM_f_multiply_constant_2_1L (x,dble(plot_x/(sx1-sx0)),23,8)
      call CM_f_multiply_constant_2_1L (y,dble(plot_y/(sy1-sy0)),23,8)
      end

      subroutine lines (x1,y1,x2,y2,color,mask)
      include '/usr/include/cm/paris-configuration-fort.h'
      include '/usr/include/cm/CMF_defs.h'
      common /display_common/ my_geometry,my_vp_set,my_color,my_display

      real x1,y1,x2,y2
      integer color
```

```
logical mask
integer entry_vp_set
integer sx3,sy3,sx4,sy4

entry_vp_set = cm_current_vp_set()
call configure (mask)
call cm_logand_context(cmf_get_field_id(mask))
sx3 = cm_allocate_stack_field(32)
sy3 = cm_allocate_stack_field(32)
sx4 = cm_allocate_stack_field(32)
sy4 = cm_allocate_stack_field(32)
call cm_u_move_1l (sx3,cmf_get_field_id(x1),32)
call cm_u_move_1l (sy3,cmf_get_field_id(y1),32)
call cm_u_move_1l (sx4,cmf_get_field_id(x2),32)
call cm_u_move_1l (sy4,cmf_get_field_id(y2),32)
call scale(sx3,sy3)
call scale(sx4,sy4)
call _plot_lines (my_color,sx3,sy3,sx4,sy4,
!            cmf_get_field_id(color))
call _refresh_fb(my_display,my_color)
call cm_deallocate_stack_through (sx3)
call cm_set_vp_set (entry_vp_set)
end

subroutine line(x1,y1,x2,y2,color)
real x1,y1,x2,y2
integer color
real cx1(1),cy1(1),cx2(1),cy2(1)
integer ccolor(1)
logical mask(1)
cx1 =x1
cy1 =y1
cx2 =x2
cy2 =y2
ccolor=color
mask=.true.
call lines(cx1,cy1,cx2,cy2,ccolor,mask)
end

subroutine circles (x,y,r,color,mask)
include '/usr/include/cm/paris-configuration-fort.h'
include '/usr/include/cm/CMF_defs.h'
common /display_common/ my_geometry,my_vp_set,my_color,my_display

real x,y,r
integer color
logical mask
integer entry_vp_set
```

```
      integer sx,sy

      entry_vp_set = cm_current_vp_set()
      call configure (mask)
      call cm_logand_context(cmf_get_field_id(mask))
      sx = cm_allocate_stack_field(32)
      sy = cm_allocate_stack_field(32)
      call cm_u_move_1l (sx,cmf_get_field_id(x),32)
      call cm_u_move_1l (sy,cmf_get_field_id(y),32)
      call scale(sx,sy)
      call _plot_circles (my_color,sx,sy,
     !            cmf_get_field_id(r), cmf_get_field_id(color))
      call _refresh_fb(my_display,my_color)
      call cm_deallocate_stack_through (sx)
      call cm_set_vp_set (entry_vp_set)
      end

      subroutine circle(x,y,r,color)
      real x,y,r
      integer color
      real cx(1),cy(1),cr(1)
      integer ccolor(1)
      logical mask(1)
      cx =x
      cy =y
      cr =r
      ccolor =color
      mask=.true.
      call circles(cx,cy,cr,ccolor,mask)
      end

      subroutine points (x,y,color,mask)
      include '/usr/include/cm/paris-configuration-fort.h'
      include '/usr/include/cm/CMF_defs.h'
      common /display_common/ my_geometry,my_vp_set,my_color,my_display

      real x,y
      integer color
      integer mask
      integer entry_vp_set
      integer sx,sy

      entry_vp_set = cm_current_vp_set()
      call configure(mask)
      sx = cm_allocate_stack_field(32)
      sy = cm_allocate_stack_field(32)
      call cm_u_move_1L (sx,cmf_get_field_id(x),32)
      call cm_u_move_1L (sy,cmf_get_field_id(y),32)
```

```
      call scale(sx,sy)
      call cm_logand_context(cmf_get_field_id(mask))
      call _plot_point (my_color,sx,sy,cmf_get_field_id(color))
      call _refresh_fb(my_display,my_color)
      call cm_deallocate_stack_through(sx)
      call cm_set_vp_set (entry_vp_set)
      end

c  should be reimplemented to use cm_u_write_to_processor
      subroutine point(x,y,color)
      real x,y
      integer color
      real cx(1),cy(1)
      integer ccolor(1),mask(1)
      cx =x
      cy =y
      ccolor =color
      mask=1
      call points(cx,cy,ccolor,mask)
      end

      subroutine label(s,llen,x,y,color)
      include '/usr/include/cm/paris-configuration-fort.h'
      include '/usr/include/cm/CMF_defs.h'
      character*255 s
      integer llen
      real x,y
      integer color
      common /display_common/ my_geometry,my_vp_set,my_color,my_display
      integer my_geometry, my_vp_set, my_color, my_display
      common /scale_common/ sx0,sy0,sx1,sy1
      real sx0,sy0,sx1,sy1
      real plot_x,plot_y

      plot_x = real(cm_geometry_axis_length(my_geometry,0))
      plot_y = real(cm_geometry_axis_length(my_geometry,1))
      if ((sx0 .eq. 0.0) .and. (sx1 .eq. 0.0) .and. (sy1 .eq. 0.0)
     !    .and. (sy0 .eq. 0.0)) then
          sx0=0.0
          sy0=0.0
          sx1=plot_x
          sy1=plot_y
          endif
      plot_x = (x-sx0)*(plot_x/(sx1-sx0))
      plot_y = (y-sy0)*(plot_y/(sy1-sy0))
      call _print_string(my_color,llen,s,plot_x,plot_y,color)
      call _refresh_fb(my_display,my_color)
      end
```

```
      subroutine set_color_value(color,r,g,b)
      integer color,r,g,b
      call set_color (color,r,g,b)
      end
```

```
#include <cm/cmfb.h>
#include <cm/paris.h>

#if defined(sparc)
# define SET_FONT_NUMBER set_font_number_
# define CLEAR clear_
# define ATTACH attach_
# define REFRESH_FB refresh_fb_
# define PLOT_POINT plot_point_
# define PLOT_LINES plot_lines_
# define PLOT_CIRCLES plot_circles_
# define DETACH detach_
# define PRINT_STRING print_string_
#endif

#define COORD_LEN 32
#define COLOR_LEN 8
#define TEXT_SPACING 5
#define lines_dimensions 32768
#define xp_size (128+COLOR_LEN)
#define circle_xp_size (96+COLOR_LEN)

#define VREF(x,y) ((CM_field_id_t)CM_add_offset_to_field_id((unsigned)x,32*y))

void _CMI_scan_with_f_add_1d ();

struct font {
char *memory;
unsigned char_x_size,char_y_size;
char *widths;
};

static float x_constant,y_constant;
static struct font *plot_font;
static CM_field_id_t constant_x;
static CM_field_id_t constant_y;
static CM_vp_set_id_t lines_vp_set=0;
static struct CM_geometry_id *lines_geometry;
```

```
extern struct font fonts[];

void CM_u_f_truncate_1_1L(source,s,e)
CM_field_id_t source;
unsigned s,e;
{
CM_field_id_t s_dest = CM_allocate_stack_field (s+e+1);

CM_s_f_truncate_2_2L(s_dest, source, s+e+1, s,e);
CM_u_move_1L (source,s_dest,s+e);
CM_deallocate_stack_through (s_dest);
}

void SET_FONT_NUMBER(size)
int *size;
{
plot_font = &(fonts[*size]);
}

void CLEAR(plot_field)
CM_field_id_t *plot_field;
{
CM_vp_set_id_t entry_vp_set = CM_current_vp_set;
CM_set_vp_set(CM_field_vp_set(*plot_field));
CM_u_move_constant_1L (*plot_field,0,COLOR_LEN);
CM_set_vp_set(entry_vp_set);
}

void ATTACH(x,y)
int *x,*y;
{
int start_font =2;
unsigned dimensions[2];

dimensions[0]=lines_dimensions;
lines_geometry =CM_create_geometry(dimensions,1);
lines_vp_set = CM_allocate_vp_set (lines_geometry);
SET_FONT_NUMBER(&start_font);
}

void check_vp_set (plot_field)
CM_field_id_t *plot_field;
{
if (lines_vp_set == 0)
ATTACH(CM_geometry_axis_length (CM_vp_set_geometry
    (CM_field_vp_set(*plot_field)),0),
   CM_geometry_axis_length (CM_vp_set_geometry
    (CM_field_vp_set(*plot_field)),1));
```

```
}

void REFRESH_FB(plot_display,plot_field)
struct CMFB_display_id **plot_display;
CM_field_id_t *plot_field;
{
CM_vp_set_id_t entry_vp_set = CM_current_vp_set;
CM_set_vp_set (CM_field_vp_set(*plot_field));
CMFB_write_preshuffled_always (*plot_display,
CMFB_spare_buffer(*plot_display),
*plot_field,0,0);
CMFB_switch_buffer(*plot_display,CMFB_spare_buffer(*plot_display));
CM_set_vp_set (entry_vp_set);
}

void PLOT_POINT(plot_field,x,y,color)
CM_field_id_t *plot_field,*x,*y,*color;
{
CM_field_id_t address = CM_allocate_stack_field (21);
CM_field_id_t entry_context = CM_allocate_stack_field(1);
CM_field_id_t tx=CM_allocate_stack_field (11);
CM_field_id_t ty=CM_allocate_stack_field (11);
unsigned plot_x = CM_geometry_axis_length (CM_vp_set_geometry
    (CM_field_vp_set(*plot_field)),0);
unsigned plot_y = CM_geometry_axis_length (CM_vp_set_geometry
    (CM_field_vp_set(*plot_field)),1);

check_vp_set (plot_field);
CM_store_context(entry_context);
CM_f_lt_constant_1L (*x,(float)plot_x,23,8);
CM_logand_context_with_test();
CM_f_ge_constant_1L (*x,0.0,23,8);
CM_logand_context_with_test();
CM_f_lt_constant_1L (*y,(float)plot_y,23,8);
CM_logand_context_with_test();
CM_f_ge_constant_1L (*y,0.0,23,8);
CM_logand_context_with_test();
CM_u_f_round_2_2L (tx,*x,11,23,8);
CM_u_f_round_2_2L (ty,*y,11,23,8);
CMFB_shuffle_from_x_y (address,tx,ty,
CM_vp_set_geometry(CM_field_vp_set(*plot_field)));
CM_send_1L (*plot_field,address,*color,COLOR_LEN,(CM_field_id_t)CM_no_field);
CM_load_context (entry_context);
CM_deallocate_stack_through (address);
}

void compute_linear_patch (dest,this_point,next_point,t)
CM_field_id_t dest,this_point,next_point,t;
```

```
{
CM_f_subtract_3_1L (dest,next_point,this_point,23,8);
CM_f_multiply_2_1L (dest,t,23,8);
CM_f_add_2_1L (dest,this_point,23,8);
}

void pixels (dest,x1,y1,x2,y2)
CM_field_id_t dest,x1,y1,x2,y2;
{
CM_field_id_t temp = CM_allocate_stack_field (32);
CM_f_subtract_3_1L (temp,x1,x2,23,8);
CM_f_subtract_3_1L (dest,y1,y2,23,8);
CM_f_abs_1_1L (temp,23,8);
CM_f_abs_1_1L (dest,23,8);
CM_f_max_2_1L (dest,temp,23,8);
CM_deallocate_stack_through (temp);
}

void PLOT_LINES(plot_field,x1,y1,x2,y2,color)
CM_field_id_t *plot_field,*x1,*y1,*x2,*y2,*color;
{
CM_vp_set_id_t entry_vp_set = CM_current_vp_set;
CM_field_id_t xp,result,temp,temp2,seg,unknown,address_temp;
CM_field_id_t entry_context=CM_allocate_stack_field (1);
CM_field_id_t address=CM_allocate_stack_field (32);
CM_field_id_t procs_needed=CM_allocate_stack_field(32);
CM_field_id_t other = CM_allocate_stack_field (32);
unsigned total_procs_needed,i,iterations;
float old_xh;

/*initialize line_vp_set*/
CM_set_vp_set (lines_vp_set);
xp=CM_allocate_stack_field (xp_size);
address_temp = VREF(xp,4);
temp=CM_allocate_stack_field (32);
temp2=CM_allocate_stack_field(32);
unknown = CM_allocate_stack_field(32);
seg=CM_allocate_stack_field (1);
CM_u_move_zero_always_1L (seg,1);

CM_set_vp_set (entry_vp_set);
CM_store_context (entry_context);
pixels(procs_needed,*x1,*y1,*x2,*y2);
_CMI_scan_with_f_add_1d (address,procs_needed,CM_send_order,23,8,
  CM_upward,CM_inclusive,CM_none,0);
total_procs_needed = CM_global_f_max_1L (address,23,8)
iterations = (unsigned) total_procs_needed/lines_dimens_ns;
CM_f_subtract_2_1L (address,procs_needed,23,8);
```

```
CM_u_f_truncate_1_1L (address,23,8);
CM_make_news_coordinate_1L (lines_geometry,other,0,address,32);
for (i=0;i<=iterations;i++){
CM_u_ge_constant_1L (address,i*lines_dimensions,32);
CM_logand_context_with_test ();
CM_u_lt_constant_1L (address,(i+1)*lines_dimensions,32);
CM_logand_context_with_test ();
CM_send_1L (xp,other,*x1,32,(CM_field_id_t)CM_no_field);
CM_send_1L (VREF(xp,1),other,*y1,32,(CM_field_id_t)CM_no_field);
CM_send_1L (VREF(xp,2),other,*x2,32,(CM_field_id_t)CM_no_field);
CM_send_1L (VREF(xp,3),other,*y2,32,(CM_field_id_t)CM_no_field);
CM_send_1L (VREF(xp,4),other,*color,COLOR_LEN,seg);
CM_set_vp_set (lines_vp_set);
CM_set_context();
CM_f_move_constant_1L (temp2,1.0,23,8);
if (i != 0) {
unsigned zero = CM_fe_make_news_coordinate(lines_geometry,0,0);
CM_clear_context ();
CM_u_write_to_processor_1L (zero,CM_context_flag,1,1);
if (CM_u_read_from_processor_1L (zero,seg,1) == 0) {
CM_f_write_to_processor_1L (zero,temp2,old_xh,23,8);
CM_get_from_news_1L (xp,xp,0,CM_downward,xp_size);
CM_u_write_to_processor_1L (zero,seg,1,1);
}
CM_set_context ();
}
CM_scan_with_copy_1L (xp,xp,0,xp_size,CM_upward,CM_inclusive,
 CM_start_bit,seg);
CM_scan_with_f_add_1L (temp,temp2,0,23,8,CM_upward,CM_inclusive,
  CM_start_bit,seg);
old_xh =CM_f_read_from_processor_1L (
CM_fe_make_news_coordinate
(lines_geometry,0,lines_dimensions-1),
temp,23,8);
if (i==iterations){
CM_my_news_coordinate_1L (temp2,0,32);
CM_u_lt_constant_1L (temp2,(unsigned)(total_procs_needed%lines_dimensions)
  ,32);
CM_logand_context_with_test ();
}
pixels (temp2,xp,VREF(xp,1),VREF(xp,2),VREF(xp,3));
CM_f_divide_2_1L (temp,temp2,23,8);
compute_linear_patch (temp2,xp,VREF(xp,2),temp);
compute_linear_patch (unknown,VREF(xp,1),VREF(xp,3),temp);
PLOT_POINT(plot_field,&temp2,&unknown,&address_temp);
CM_u_move_zero_always_1L (seg,1);
CM_set_vp_set (entry_vp_set);
CM_load_context (entry_context);
```

```
}
}

void PLOT_CIRCLES(plot_field,x,y,r,color)
CM_field_id_t *plot_field,*x,*y,*r,*color;
{
CM_vp_set_id_t entry_vp_set = CM_current_vp_set;
CM_field_id_t xp,result,temp,temp2,seg,unknown;
CM_field_id_t entry_context=CM_allocate_stack_field (1);
CM_field_id_t address=CM_allocate_stack_field (32);
CM_field_id_t procs_needed=CM_allocate_stack_field(32);
CM_field_id_t other = CM_allocate_stack_field (32);
unsigned total_procs_needed,i,iterations;
float old_xh;

check_vp_set( plot_field);
/*initialize line_vp_set*/
CM_set_vp_set (lines_vp_set);
xp=CM_allocate_stack_field (circle_xp_size);
temp=CM_allocate_stack_field (32);
temp2=CM_allocate_stack_field(32);
unknown = VREF(xp,3);
seg=CM_allocate_stack_field (1);
CM_u_move_zero_always_1L (seg,1);

CM_set_vp_set (entry_vp_set);
CM_store_context (entry_context);
CM_f_multiply_constant_3_1L (procs_needed,*r,3.1415926,23,8);
_CMI_scan_with_f_add_1d (address,procs_needed,CM_send_order,23,8,
  CM_upward,CM_inclusive,CM_none,0);
total_procs_needed = CM_global_f_max_1L (address,23,8);
iterations = (unsigned) total_procs_needed/lines_dimensions;
CM_f_subtract_2_1L (address,procs_needed,23,8);
CM_u_f_truncate_1_1L (address,23,8);
CM_make_news_coordinate_1L (lines_geometry,other,0,address,32);
for (i=0;i<=iterations;i++){
CM_u_ge_constant_1L (address,i*lines_dimensions,32);
CM_logand_context_with_test ();
CM_u_lt_constant_1L (address,(i+1)*lines_dimensions,32);
CM_logand_context_with_test ();
CM_send_1L (xp,other,*x,32,(CM_field_id_t)CM_no_field);
CM_send_1L (VREF(xp,1),other,*y,32,(CM_field_id_t)CM_no_field);
CM_send_1L (VREF(xp,2),other,*r,32,(CM_field_id_t)CM_no_field);
CM_send_1L (VREF(xp,3),other,*color,COLOR_LEN,seg);
CM_set_vp_set (lines_vp_set);
CM_set_context();
CM_f_move_constant_1L (temp2,1.0,23,8);
if (i != 0) {
```

```
unsigned zero = CM_fe_make_news_coordinate(lines_geometry,0,0);
CM_clear_context ();
CM_u_write_to_processor_1L (zero,CM_context_flag,1,1);
if (CM_u_read_from_processor_1L (zero,seg,1) == 0) {
CM_f_write_to_processor_1L (zero,temp2,old_xh,23,8);
CM_get_from_news_1L (xp,xp,0,CM_downward,circle_xp_size);
CM_u_write_to_processor_1L (zero,seg,1,1);
}
CM_set_context ();
}
CM_scan_with_copy_1L (xp,xp,0,circle_xp_size,CM_upward,CM_inclusive,
 CM_start_bit,seg);
CM_scan_with_f_add_1L (temp,temp2,0,23,8,CM_upward,CM_inclusive,
  CM_start_bit,seg);
old_xh =CM_f_read_from_processor_1L (
CM_fe_make_news_coordinate
(lines_geometry,0,lines_dimensions-1),
temp,23,8);
if (i==iterations){
CM_my_news_coordinate_1L (temp2,0,32);
CM_u_lt_constant_1L (temp2,(unsigned)(total_procs_needed%lines_dimensions)
  ,32);
CM_logand_context_with_test ();
}
CM_f_multiply_constant_3_1L (temp2,VREF(xp,2),3.1415926,23,8);
CM_f_divide_2_1L (temp,temp2,23,8);
CM_f_multiply_constant_2_1L (temp,2*3.1415926,23,8);
CM_f_sin_2_1L (temp2,temp,23,8);
CM_f_cos_1_1L (temp,23,8);
CM_f_multiply_2_1L (temp2,VREF(xp,2),23,8);
CM_f_multiply_2_1L (temp,VREF(xp,2),23,8);
CM_f_add_2_1L (temp2,xp,23,8);
CM_f_add_2_1L (temp,VREF(xp,1),23,8);
PLOT_POINT(plot_field,&temp2,&temp,&unknown);
CM_u_move_zero_always_1L (seg,1);
CM_set_vp_set (entry_vp_set);
CM_load_context (entry_context);
}
CM_deallocate_stack_through (entry_context);
}

void DETACH()
{
CM_deallocate_vp_set (lines_vp_set);
CM_deallocate_geometry (lines_geometry);
}

unsigned font_value (char_value,x,y)
```

```
unsigned char_value,x,y;
{
unsigned index=(char_value-32)*plot_font->char_x_size+x+
(y*plot_font->char_x_size*96);
unsigned byte = index/8;
unsigned bit = index%8;
unsigned char result =plot_font->memory[byte];
result = (result >> bit) & 1;
return result;
}

#if defined(sparc)
struct ftn_string {char str[255]};
#else
struct ftn_string {short len; char * str};
#endif

void PRINT_STRING(plot_field,length,string,xp,yp,color)
     CM_field_id_t *plot_field;
     int *length;
     struct ftn_string *string;
     float *xp,*yp;
     int *color;
{
  unsigned plot_x = CM_geometry_axis_length (CM_vp_set_geometry
     (CM_field_vp_set(*plot_field)),0);
  unsigned plot_y = CM_geometry_axis_length (CM_vp_set_geometry
     (CM_field_vp_set(*plot_field)),1);
  int k=0,temp_x_constant=(int)(*xp),xd,yd,i,x,y,c;
  unsigned address;
  CM_vp_set_id_t entry_vp_set = CM_current_vp_set;

  check_vp_set (plot_field);
  CM_set_vp_set (CM_field_vp_set(*plot_field));
  for (i=0;i<*length;i++){
    c=string->str[i];
    for(x=0;x<(plot_font->widths)[c-32];x++)
      for (y=0;y<(plot_font->char_y_size);y++){
        xd = x+temp_x_constant;
        yd = y+(int)(*yp);
if (font_value(c,x,y)&&(xd>0)&&(yd>0)&&(xd<plot_x)&&(yd<plot_y)){
          address = CMFB_fe_shuffle_from_x_y (xd,yd,
      CM_vp_set_geometry
      (CM_field_vp_set(*plot_field)));
          CM_u_write_to_processor_1L(address,*plot_field,*color,COLOR_LEN);
}
      }
    temp_x_constant += x;
```

```
  }
  CM_set_vp_set(entry_vp_set);
}
```

B9 Surface Routines

```
c
c     this is a general purpose routine for using
c     paris from cmf. it sets vp set and the context
c     to reflect the processor configuration in array 'x'
c
      subroutine configure(x)
      integer x
      integer address,i
      include '/usr/include/cm/paris-configuration-fort.h'
      include '/usr/include/cm/CMF_defs.h'

      call cm_set_vp_set (cmf_get_vp_set_id(x))
      call cm_set_context()
      address = cm_allocate_stack_field(32)
      call cm_my_news_coordinate_1l (address,0,32)
      call cm_u_eq_constant_1l(address,0,32)
      call cm_logand_context_with_test()
      do i=1,cmf_get_rank(x)
         call cm_my_news_coordinate_1l (address,i,32)
         call cm_u_lt_constant_1l(address,cmf_get_axis_extent(x,i-1),32)
         call cm_logand_context_with_test()
      enddo
      call cm_deallocate_stack_through (address)
      end

      subroutine surface (z,color,theta,phi)
      real z,theta,phi
      integer color
      include '/usr/include/cm/paris-configuration-fort.h'
      include '/usr/include/cm/CMF_defs.h'
      common /display_common/ my_geometry, my_vp_set, my_color,
     +     my_display
      integer my_geometry, my_vp_set, my_color, my_display

      call configure(z)
      call _surface_internal (my_color,cmf_get_field_id(z),
     &                        cmf_get_field_id(color),
     &                        theta,phi,1,my_display)
      return
      end

      subroutine surface_over (z,color,theta,phi)
      real z,theta,phi
      integer color
      include '/usr/include/cm/paris-configuration-fort.h'
```

```
      include '/usr/include/cm/CMF_defs.h'
      common /display_common/ my_geometry, my_vp_set, my_color,
     +     my_display
      integer my_geometry, my_vp_set, my_color, my_display


      call configure(z)
      call _surface_internal (my_color,cmf_get_field_id(z),
     &                        cmf_get_field_id(color),
     &                        theta,phi,0,my_display)
      return
      end

      subroutine shade (dest,z,lx,ly,lz)
      integer dest
      real x,y,z,lx,ly,lz
      include '/usr/include/cm/paris-configuration-fort.h'
      include '/usr/include/cm/CMF_defs.h'
      call configure (dest)
      call _light_intensity (cmf_get_field_id(dest),
     &                       cmf_get_field_id(z),lx,ly,lz)
      return
      end


#include <cm/paris.h>
#include <cm/cmfb.h>
#include <cm/cmfs.h>
#include <cm/cm_file.h>
#include <math.h>
#include <stdio.h>

#if defined(sparc)
# define LIGHT_INTENSITY light_intensity_
# define SURFACE_INTERNAL surface_internal_
#endif

#define AMBIENT 60
#define COLOR_DEPTH 8
void rotate_constant (x,y,angle)
   CM_field_id_t x,y;
   float angle;
{
 CM_field_id_t temp = CM_allocate_stack_field (32);
 float s=sin(angle),c=cos(angle);

 CM_u_move_1L (temp,x,32);
 CM_f_multiply_constant_2_1L (x,c,23,8);
```

```
 CM_f_mult_const_add_1L (x,y,(-s),x,23,8);
 CM_f_multiply_constant_2_1L (y,c,23,8);
 CM_f_mult_const_add_1L (y,temp,s,y,23,8);
 CM_deallocate_stack_through (temp);
}


void SURFACE_INTERNAL (dest,z,color,theta,phi,erase_p,display)
     CM_field_id_t *dest,*z,*color;
     float *theta,*phi;
     unsigned *erase_p;
     struct CMFB_display_id **display;
{
  CM_field_id_t x,y,shade,temp,address,message,buffer,seg,mz;
  CM_vp_set_id_t o_vp_set = CM_current_vp_set;
  CM_vp_set_id_t b_vp_set = CM_field_vp_set(*dest);
  unsigned dimensions[3],buffer_d[2],zoom;
  unsigned message_length,coord_length;

  dimensions[0]=1;
  dimensions[1]=CM_geometry_axis_length(CM_vp_set_geometry(o_vp_set),1);
  dimensions[2]=CM_geometry_axis_length(CM_vp_set_geometry(o_vp_set),2);
  buffer_d[0]=CM_geometry_axis_length(CM_vp_set_geometry(b_vp_set),0);
  buffer_d[1]=CM_geometry_axis_length(CM_vp_set_geometry(b_vp_set),1);
  zoom = 1024/buffer_d[0]-1;

  if ((buffer_d[0] != dimensions[1]*2)||
      (buffer_d[1] != dimensions[2]*2)){
    fprintf (stderr,"surface dimensions are incompatible with framebuffer");
    exit(1);
  }
  coord_length=CM_geometry_coordinate_length (CM_vp_set_geometry(b_vp_set),1);
  message_length = COLOR_DEPTH + coord_length;
  message = CM_allocate_stack_field (message_length);
  x = CM_allocate_stack_field(32);
  y = CM_allocate_stack_field(32);
  mz = CM_allocate_stack_field(32);
  shade = CM_allocate_stack_field(32);
  address = CM_allocate_stack_field(32);
  seg = CM_allocate_stack_field(1);

  CM_set_vp_set (b_vp_set);
  buffer = CM_allocate_stack_field(message_length);
  CM_u_move_zero_always_1L (buffer,message_length);
  if (*erase_p==1)
    CM_u_move_zero_always_1L (*dest,8);

  CM_set_vp_set(o_vp_set);
```

```
  CM_u_move_1L (mz,*z,32);
  CM_u_move_zero_1L (seg,1);
  CM_my_news_coordinate_1L (shade,1,32);
  CM_u_le_constant_1L (shade,3,32);
  CM_logior_2_1L (seg,CM_test_flag,1);
  CM_u_ge_constant_1L (shade,dimensions[1]-3,32);
  CM_logior_2_1L (seg,CM_test_flag,1);
  CM_f_u_float_2_2L (x,shade,32,23,8);

  CM_my_news_coordinate_1L (shade,2,32);
  CM_u_move_1L (CM_add_offset_to_field_id(message,COLOR_DEPTH),
shade,coord_length);
  CM_u_le_constant_1L (shade,3,32);
  CM_logior_2_1L (seg,CM_test_flag,1);
  CM_u_ge_constant_1L (shade,dimensions[2]-3,32);
  CM_logior_2_1L (seg,CM_test_flag,1);
  CM_f_u_float_2_2L (y,shade,32,23,8);

  CM_f_subtract_constant_2_1L (x,(float)dimensions[1]/2.0,23,8);
  CM_f_subtract_constant_2_1L (y,(float)dimensions[2]/2.0,23,8);
  CM_u_move_1L (message,*color,8);
  CM_load_context(seg);
  CM_u_move_zero_1L (message,COLOR_DEPTH);
  CM_set_context();
  rotate_constant (x,y,-(*theta));
  CM_f_add_constant_3_1L (shade,y,(float)buffer_d[1]/2.0,23,8);
  CM_u_f_round_2_2L (message+COLOR_DEPTH,shade,coord_length,23,8);
  rotate_constant (y,mz,*phi);
  CM_f_add_constant_2_1L (x,(float)buffer_d[0]/2.0,23,8);
  CM_f_add_constant_2_1L (y,(float)buffer_d[1]/2.0,23,8);
  CM_u_f_round_2_2L (shade,x,coord_length,23,8);
  CM_make_news_coordinate_1L (CM_vp_set_geometry(b_vp_set),address,0,
      shade,coord_length);
  CM_u_f_round_2_2L (shade,y,coord_length,23,8);
  CM_deposit_news_coordinate_1L (CM_vp_set_geometry(b_vp_set),address,1,
 shade,coord_length);
  CM_send_with_u_max_1L(buffer,address,message,message_length,CM_no_field);
  CM_set_vp_set(b_vp_set);
  CM_scan_with_u_max_1L (buffer,buffer,1,message_length,
 CM_upward, CM_inclusive,CM_none,CM_no_field);
  CM_u_gt_zero_1L(buffer,8);
  CM_logand_context_with_test();
  CMFB_preshuffle_for_write (*dest,buffer,8);
  CMFB_write_preshuffled_always (*display,CMFB_spare_buffer(*display),
 *dest,0,0);
  CM_set_context();
  CMFB_switch_buffer(*display,CMFB_spare_buffer(*display));
  CM_deallocate_stack_through (message);
```

```
}


void smooth_float_image (image)
  CM_field_id_t image;
{
 CM_field_id_t temp = CM_allocate_stack_field(32);
 CM_field_id_t temp2 = CM_allocate_stack_field(32);
 CM_f_move_zero_1L (temp,23,8);

 CM_f_news_add_always_2_1L (temp,image,0,CM_upward,23,8);
 CM_f_news_add_always_2_1L (temp,image,1,CM_upward,23,8);
 CM_f_news_add_always_2_1L (temp,image,0,CM_downward,23,8);
 CM_f_news_add_always_2_1L (temp,image,1,CM_downward,23,8);
 CM_f_add_2_1L (image,temp,23,8);
 CM_f_divide_constant_2_1L (image,5.0,23,8);
 CM_deallocate_stack_through (temp);
}


void LIGHT_INTENSITY (dest,oz,rz,rx)
     CM_field_id_t *dest,*oz;
     float *rz,*rx;
{
  CM_field_id_t gx = CM_allocate_stack_field(96);
  CM_field_id_t gy = CM_add_offset_to_field_id(gx,32);
  CM_field_id_t gz = CM_add_offset_to_field_id(gy,32);
  CM_field_id_t x = CM_allocate_stack_field(32);
  CM_field_id_t y = CM_allocate_stack_field(32);
  CM_field_id_t z = CM_allocate_stack_field(32);
  CM_field_id_t norm = CM_allocate_stack_field(32);
  float min,max;

  CM_my_news_coordinate_1L (z,1,32);
  CM_f_u_float_2_2L (x,z,32,23,8);
  CM_f_subtract_constant_2_1L
    (x,(float)CM_geometry_axis_length
     (CM_vp_set_geometry(CM_current_vp_set),1)/2.0,23,8);
  CM_my_news_coordinate_1L (z,2,32);
  CM_f_u_float_2_2L (y,z,32,23,8);
  CM_f_subtract_constant_2_1L
    (y,(float)CM_geometry_axis_length
     (CM_vp_set_geometry(CM_current_vp_set),2)/2.0,23,8);

  CM_u_move_1L (z,*oz,32);
  rotate_constant (x,y,-(*rz));
  rotate_constant (y,z,*rx);
  CM_f_news_sub_always_3_1L (gx,y,y,1,CM_upward,23,8);
```

```
CM_f_news_sub_always_3_1L (gy,x,x,1,CM_upward,23,8);
CM_f_news_sub_always_3_1L (gz,x,x,1,CM_upward,23,8);
CM_f_news_sub_mult_4_1L (gx,z,z,gx,2,CM_upward,23,8);
CM_f_news_sub_mult_4_1L (gy,z,z,gy,2,CM_upward,23,8);
CM_f_news_sub_mult_4_1L (gz,y,y,gz,2,CM_upward,23,8);
CM_f_news_sub_always_3_1L (*dest,z,z,1,CM_upward,23,8);
CM_f_news_sub_mult_4_1L (*dest,y,y,*dest,2,CM_upward,23,8);
CM_f_subtract_2_1L (gx,*dest,23,8);
CM_f_news_sub_always_3_1L (*dest,z,z,1,CM_upward,23,8);
CM_f_news_sub_mult_4_1L (*dest,x,x,*dest,2,CM_upward,23,8);
CM_f_subtract_2_1L (gy,*dest,23,8);
CM_f_news_sub_always_3_1L (*dest,y,y,1,CM_upward,23,8);
CM_f_news_sub_mult_4_1L (*dest,x,x,*dest,2,CM_upward,23,8);
CM_f_subtract_2_1L (gz,*dest,23,8);
CM_f_negate_1_1L (gy,23,8);
/* normal of cross product*/
CM_f_multiply_3_1L (norm,gx,gx,23,8);
CM_f_mult_add_1L (norm,gy,gy,norm,23,8);
CM_f_mult_add_1L (norm,gz,gz,norm,23,8);

/* perform dot*/
CM_f_sub_const_mult_1L (gx,x,1000.0,gx,23,8);
CM_f_sub_const_mult_1L (gy,y,1000.0,gy,23,8);
CM_f_sub_const_mult_1L (gz,z,-1000.0,gz,23,8);
CM_f_add_3_1L (*dest,gx,gy,23,8);
CM_f_add_2_1L (*dest,gz,23,8);

/* normal of light vector (can be combined)*/
CM_f_subtract_constant_3_1L (gx,x,1000.0,23,8);
CM_f_subtract_constant_3_1L (gy,y,1000.0,23,8);
CM_f_subtract_constant_3_1L (gz,z,-1000.0,23,8);
CM_f_multiply_2_1L (gx,gx,23,8);
CM_f_mult_add_1L (gx,gy,gy,gx,23,8);
CM_f_mult_add_1L (gx,gz,gz,gx,23,8);
CM_f_multiply_2_1L (norm,gx,23,8);

CM_f_sqrt_1_1L (norm,23,8);
CM_f_divide_2_1L (*dest,norm,23,8);
smooth_float_image(*dest);
smooth_float_image(*dest);
smooth_float_image(*dest);
smooth_float_image(*dest);
CM_store_context(gx);
CM_f_lt_zero_1L (*dest,23,8);
CM_logand_context_with_test();
CM_f_move_zero_1L (*dest,23,8);
CM_load_context(gx);
CM_f_multiply_constant_3_1L (norm,*dest,255.0-(float)AMBIENT,23,8);
```

```
  CM_u_f_truncate_2_2L (*dest,norm,8,23,8);
  CM_u_add_constant_2_1L (*dest,AMBIENT,8);
  CM_deallocate_stack_through (gx);
}
```

B10 Polynomial Evaluation Routines

```
      INTEGER FUNCTION MAKE_HORNER_COEF(COEF_ARRAY, LENGTH)
      INTEGER COEF_ARRAY, LENGTH

      INCLUDE '/usr/include/cm/paris-configuration-fort.h'
      INCLUDE '/usr/include/cm/CMF_defs.h'

      INTEGER TEMP_LUT
      INTEGER RESULT
      INTEGER MAKE_REAL_LOOKUP

      TEMP_LUT = MAKE_REAL_LOOKUP(COEF_ARRAY, LENGTH)
      CALL _MAKE_HORNER_COEF(RESULT, TEMP_LUT, COEF_ARRAY, LENGTH)
      MAKE_HORNER_COEF = RESULT

      END FUNCTION MAKE_HORNER_COEF


      SUBROUTINE EVAL_HORNER(RESULT, COEFS, X)
      INTEGER RESULT, COEFS, X

      INCLUDE '/usr/include/cm/paris-configuration-fort.h'
      INCLUDE '/usr/include/cm/CMF_defs.h'

      integer the_vp_set
      the_vp_set = CMF_get_vp_set_id(x)
      call CM_set_vp_set(the_vp_set)
      CALL _EVAL_HORNER(CMF_GET_FIELD_ID(RESULT), COEFS,
     +      CMF_GET_FIELD_ID(X))

      CALL CMF_set_is_modified(result,MODIF)
      RETURN
      END



      SUBROUTINE FREE_HORNER_COEF(COEFS)
      INTEGER COEFS

      CALL _FREE_HORNER_COEF(COEFS)

      RETURN
      END
```

```
#include <stdio.h>
#include <cm/paris.h>
#include "../lookup/lookup.h"
#include <cm/impctl.h>
struct horner_coef {struct lut * mylut; int size;};

typedef struct horner_coef horner_coef_t;

void make_horner_coef();
void eval_horner();
void free_horner_coef();

#if defined(sparc)
# define MAKE_HORNER_COEF make_horner_coef_
# define EVAL_HORNER eval_horner_
# define FREE_HORNER_COEF free_horner_coef_
# define FREE_LOOKUP free_lookup_
#endif

IMP_impid_t coefs_imp;

char *malloc();

void MAKE_HORNER_COEF(horner_lut, temp_lut, coef_array, length)
  horner_coef_t **horner_lut;
  lut_t **temp_lut;
  float *coef_array;
  int *length;
  {
#ifndef ILB
  coefs_imp = IMP_open_imp("fast-poly.imi", "CMISPOLYALWAYS", NO_LOAD_IMPS);
#else
  IMP_include_imp_library(ILBNAME);
  coefs_imp = IMP_open_imp(IMP_LIBRARIES, "CMISPOLYALWAYS", NO_LOAD_IMPS);
#endif
  *horner_lut = (horner_coef_t *) malloc(sizeof(horner_coef_t));
  (**horner_lut).mylut = *temp_lut;
  (**horner_lut).size = *length;
  if (*length < 3)
    printf("Error: polynomial must have at least 3 coeficients\n");
}


void eval_horner_internal(result, coefs, x, size)
  CM_field_id_t result, coefs, x;
  int size;
 {
```

```
  unsigned x_addr, result_addr, coefs_addr;
  unsigned x_loc, x_inc, result_loc, result_inc, coefs_loc;
  _CMI_decode_location_increment(result, result_loc, result_inc);
  _CMI_decode_location_increment(x, x_loc, x_inc);
  _CMI_decode_location(coefs, coefs_loc);
  IMP_execute_imp_id(coefs_imp);
  IMP_send_imp_data(size-2);
  IMP_send_imp_data(result_loc);
  IMP_send_imp_data(result_inc);
  IMP_send_imp_data(x_loc);
  IMP_send_imp_data(x_inc);
  IMP_send_imp_data(coefs_loc);
}

void EVAL_HORNER(result, coefs, x)
  CM_field_id_t *result;
  horner_coef_t **coefs;
  CM_field_id_t *x;
 {
  if ((**coefs).mylut->allocated_p)
    eval_horner_internal(*result, (**coefs).mylut->cm_field, *x, (**coefs).size);
  else {
    printf("eval_horner: coeficients deallocated!  returning 0.\n");
    *result = 0.0;
  }
}

void FREE_HORNER_COEF(coefs)
 horner_coef_t **coefs;
 {
  FREE_LOOKUP(*coefs);
 }
```

B11 Fast Fourier Transform Routine

```
      subroutine fft(re_dest,im_dest,re_source,im_source,
     +     operation)

c    re_dest      : single precision real destination field
c    im_dest      : single precision imaginary destination field
c    re_source    : single precision real source field
c    im_source    : single precision imaginary source field
c    operation    : 0 no operation
c                   1 forward transform
c                   2 inverse transform

      include '/usr/include/cm/paris-configuration-fort.h'
      include '/usr/include/cm/CMF_defs.h'

      integer re_dest,im_dest,re_source,im_source,operation
      integer re_dest_id,im_dest_id,re_source_id,im_source_id
      integer vpset

      re_dest_id = cmf_get_field_id(re_dest)
      im_dest_id = cmf_get_field_id(im_dest)
      re_source_id = cmf_get_field_id(re_source)
      im_source_id = cmf_get_field_id(im_source)


      vpset = cmf_get_vp_set_id(re_dest)
      if (.not. ( (vpset .eq. cmf_get_vp_set_id(im_dest)) .and.
     +     (vpset .eq. cmf_get_vp_set_id(re_source)) .and.
     +     (vpset .eq. cmf_get_vp_set_id(im_source)))) then
        print*,' ERROR source and dest are not in the same vpset'
        stop
      endif

c call c routine which makes paris call to fft

      call _CFFT(vpset,re_dest_id,im_dest_id,re_source_id,
     +           im_source_id,operation)

      call CMF_set_is_modified(re_dest,MODIF)
      call CMF_set_is_modified(im_dest,MODIF)
      return
      end

#define len 32
#define axesmax 31
#define n 4
```

```
#include <cm/paris.h>
#include <stdio.h>

#if defined(sparc)
# define CFFT cfft_
#endif

static CM_geometry_id_t fft_geo[n];
static CMSSL_fft_setup_t fft_init[n];
static int setup_ptr = 0;
static int deallocate = 0;
extern CMSSL_fft_setup_t CMSSL_c_fft_setup();


void CFFT(vpset,re_dest,im_dest,re_source,im_source,operation)

     CM_vp_set_id_t *vpset;
     CM_field_id_t *re_dest,*im_dest,*re_source,*im_source;
     int operation[];


{ CM_field_id_t dest_complex_field,source_complex_field;

  int ops[axesmax],source_bit_order[axesmax],dest_bit_order[axesmax];
  int source_cm_order[axesmax],dest_cm_order[axesmax],scale[axesmax];
  int rank,i,imag_part;
  CM_geometry_id_t geom_id;
  int index;

  CM_set_vp_set(*vpset);
  CM_set_context();

  geom_id = CM_vp_set_geometry(*vpset);
  rank = CM_geometry_rank(geom_id);

  source_complex_field = CM_allocate_stack_field(2 * len);
  dest_complex_field = CM_allocate_stack_field(2 * len);
  imag_part = CM_add_offset_to_field_id(source_complex_field,len);
  CM_u_move_1L(source_complex_field,*re_source,len);
  CM_u_move_1L(imag_part,*im_source,len);

/*  set up defaults for call to paris function c-fft1

cm order:

            CMSSL_default 0
            CMSSL_send 1
            CMSSL_news 2
```

```
bit order:

            CMSSL_normal 0
            CMSSL_bit_reversed 1

scale:

            CMSSL_noscale 0
            CMSSL_scale_sqrt 1
            CMSSL_scale_n 2      */


  for (i=1;i<rank;i++)
    {if (operation[i-1] == 0)
       {ops[i] = CMSSL_nop;
source_bit_order[i] = CMSSL_normal;
dest_bit_order[i] = CMSSL_normal;
      };
     if (operation[i-1] == 1)
       {ops[i] = CMSSL_f_xform;
source_bit_order[i] = CMSSL_normal;
dest_bit_order[i] = CMSSL_bit_reversed;
      };
     if (operation[i-1] == 2)
       {ops[i] = CMSSL_i_xform;
source_bit_order[i] = CMSSL_bit_reversed;
dest_bit_order[i] = CMSSL_normal;
      };
      source_cm_order[i] = CMSSL_default_124;
      dest_cm_order[i] = CMSSL_default_124;
      scale[i] = CMSSL_noscale;
   }

/*    no operation along axis 0   */


  ops[0] = CMSSL_nop;
  source_cm_order[0] = CMSSL_default_124;
  dest_cm_order[0] = CMSSL_default_124;
  scale[0] = CMSSL_noscale;

/*    check if the front end setup descriptor for this particular
      geometry has already been allocated and is contained
      in the current list of setup_id's (if not then add to the list).
*/

  index = -1;
```

```
  for (i=0;i<n && index == -1;i++)
      if (geom_id == fft_geo[i]) index = i;

  if (index == -1)

/* add to list and deallocate a setup if out of room */

    {  if (deallocate == 1)
   CMSSL_deallocate_setup(fft_init[setup_ptr]);
       fft_geo[setup_ptr] = geom_id;
/* create setup descriptor for this geometry */
       fft_init[setup_ptr] = CMSSL_c_fft_setup(geom_id);
       index = setup_ptr;
       setup_ptr++;
/* reset setup_ptr to zero if end of list is reached */
       if (setup_ptr == n)
 {
   setup_ptr = 0;
           deallocate = 1;
 }
     };

/*  **** paris call  */

  CMSSL_c_c_fft(dest_complex_field,source_complex_field,
fft_init[index],ops,source_bit_order,
dest_bit_order,source_cm_order,dest_cm_order,scale);


/*  extract real and imaginary parts */

  imag_part = CM_add_offset_to_field_id(dest_complex_field,len);
  CM_u_move_1L(*re_dest,dest_complex_field,len);
  CM_u_move_1L(*im_dest,imag_part,len);

  CM_deallocate_stack_through(source_complex_field);
}
```

B12 Matrix Multiply Routine

```
      subroutine matmul1(x,y,z)

c   parameters:
c
c     x  : single precision real array; source1
c     y  : single precision real array; source2
c     z  : single precision real array; result of matrix multiply
c                                       of x and y

      include '/usr/include/cm/paris-configuration-fort.h'
      include '/usr/include/cm/CMF_defs.h'

      integer x,x_id,x_geom,x_vpset
      integer y,y_id,y_geom,y_vpset
      integer z,z_id,z_geom,z_vpset
      integer a_geom,b_geom,c_geom
      integer rank,descriptor_array(7,7)
      integer i

      x_id = cmf_get_field_id(x)
      y_id = cmf_get_field_id(y)
      z_id = cmf_get_field_id(z)

      x_vpset = cmf_get_vp_set_id(x)
      y_vpset = cmf_get_vp_set_id(y)
      z_vpset = cmf_get_vp_set_id(z)

      if (x_id .eq. 0) then
         print *,
     +'Error, the a argument to matmul1  is not on the CM'
         stop
      endif

      if (y_id .eq. 0) then
         print *,
     +'Error, the b argument to matmul1 is not on the CM'
         stop
      endif

      if (z_id .eq. 0) then
         print *,
     +'Error, the c argument to matmul1 is not on the CM'
         stop
      endif
```

```
 if (cmf_get_axis_extent(x,1) .ne. cmf_get_axis_extent(y,0) .or.
+    cmf_get_axis_extent(z,0) .ne. cmf_get_axis_extent(x,0) .or.
+    cmf_get_axis_extent(z,1) .ne. cmf_get_axis_extent(y,1)) then
    print*,
+'Error, the dimensions are not compatible'
    stop
 endif

 x_geom = cm_vp_set_geometry(x_vpset)
 rank = cm_geometry_rank(x_geom)

 do i=1,rank-1
    descriptor_array(1,i) = CM_geometry_axis_length(x_geom,i)
    descriptor_array(3,i) = CM_geometry_axis_ordering(x_geom,i)
    descriptor_array(4,i) =
+        CM_geometry_axis_on_chip_bits(x_geom,i)
    descriptor_array(6,i) =
+        CM_geometry_axis_off_chip_bits(x_geom,i)
 enddo

 a_geom = cm_create_detailed_geometry(descriptor_array,rank-1)

 call cm_set_vp_set_geometry(x_vpset,a_geom)

 if (x_vpset .ne. y_vpset) then

 y_geom = cm_vp_set_geometry(y_vpset)
 rank = cm_geometry_rank(y_geom)

 do i=1,rank-1
    descriptor_array(1,i) = CM_geometry_axis_length(y_geom,i)
    descriptor_array(3,i) = CM_geometry_axis_ordering(y_geom,i)
    descriptor_array(4,i) =
+        CM_geometry_axis_on_chip_bits(y_geom,i)
    descriptor_array(6,i) =
+        CM_geometry_axis_off_chip_bits(y_geom,i)
 enddo
 b_geom = cm_create_detailed_geometry(descriptor_array,rank-1)
 call cm_set_vp_set_geometry(y_vpset,b_geom)

 endif

 if (x_vpset .ne. z_vpset .and. y_vpset .ne. z_vpset) then

 z_geom = cm_vp_set_geometry(z_vpset)
 rank = cm_geometry_rank(z_geom)

 do i=1,rank-1
```

```
      descriptor_array(1,i) = CM_geometry_axis_length(z_geom,i)
      descriptor_array(3,i) = CM_geometry_axis_ordering(z_geom,i)
      descriptor_array(4,i) =
     +        CM_geometry_axis_on_chip_bits(z_geom,i)
      descriptor_array(6,i) =
     +        CM_geometry_axis_off_chip_bits(z_geom,i)
      enddo

      c_geom = cm_create_detailed_geometry(descriptor_array,rank-1)
      call cm_set_vp_set_geometry(z_vpset,c_geom)

      endif

      call cm_u_move_zero_always_1l(z_id,32)
      call cmssl_s_matrix_multiply(x_id,y_id,z_id)

      call cm_set_vp_set_geometry(x_vpset, x_geom)
      call cm_deallocate_geometry(a_geom)
      if (x_vpset .ne. y_vpset) then
        call cm_set_vp_set_geometry(y_vpset, y_geom)
        call cm_deallocate_geometry(b_geom)
      endif
      if (x_vpset .ne. z_vpset .and. y_vpset .ne. z_vpset) then
        call cm_set_vp_set_geometry(z_vpset, z_geom)
        call cm_deallocate_geometry(c_geom)
      endif

      return
      end
```

B13 *Linear System Routines*

```
      subroutine gauss(ndim,mat)

c   solve ndim by ndim system of linear equations
c   forcing vector is augmented (last column of mat)

c
c     parameters :  ndim - input; integer;  dimension of system
c                   mat - input; real;
c                          ndim by ndim+1 system of linear
c                          equations to be solved.  Forcing
c                          vector is augmented (last column).
c                         output; solution vector is contained
c                                 in last column
c

      integer ndim,i
      real mat(ndim,ndim+1),temp(ndim+1)
      logical mask(ndim,ndim+1)
      logical find_index

cmf$  layout mat(:news,:news),temp(:news),mask(:news,:news)

      mask = .false.

      do i=1,ndim

c   re-select grid each time through

         mask = .true.

c   select maximum column element from rows i,i+1,...,ndim
c   for pivotting and swap rows

c ******      j = maxloc(abs(mat(i:ndim,i))) + (i-1)
c   maxloc is not working in current release 0.5 of compiler

c   swap rows

c        temp = mat(i,:)
c        mat(i,:) = mat(j,:)
c        mat(j,:) = temp

c   divide row i by pivot

        mat(i,:) = mat(i,:)/mat(i,i)
```

```
c   current row i is left unchanged (masked off)
c   subtract multiples of it from rows i+1,i+2,...,ndim
c   to zero column i

        mask(i,:) = .false.
        where (mask)
           mat = mat - spread(mat(i,:),1,ndim) *
     +             spread(mat(:,i),2,ndim+1)
        endwhere
      enddo
      return
      end

      subroutine inv(ndim,mat)

c   solve for inverse of ndim by ndim system

c
c     parameters :  ndim - input; integer;  dimension of system
c                   mat - input; real;
c                          ndim by ndim system
c                         output; inverse of system returned
c                                 overwrites mat
c

      integer ndim,i
      real mat(ndim,ndim)
cmf$  layout mat(:news,:news)
      real mattemp(ndim,2*ndim),temp(2*ndim)
cmf$  layout mattemp(:news,:news),temp(:news)
      logical mask(ndim,2*ndim)
cmf$  layout mask(:news,:news)
      logical find_index

c  intialize mattemp rectangular   [mat|identity]

      mattemp = 0.0
      do i=1,ndim
         mattemp(i,ndim+i) = 1.0
      enddo
      mattemp(1:ndim,1:ndim) = mat(1:ndim,1:ndim)

      mask = .false.
      do i=1,ndim

c   re-select grid each time through
```

```
          mask = .true.

c    select maximum column element from rows i,i+1,...,ndim
c    for pivotting and swap rows

c ******      j = maxloc(abs(mattemp(i:ndim,i))) + (i-1)
c    swap rows

c         temp = mattemp(i,:)
c         mattemp(i,:) = mattemp(j,:)
c         mattemp(j,:) = temp

c    divide row i by pivot

         mattemp(i,:) = mattemp(i,:)/mattemp(i,i)

c    current row i is left unchanged (masked off)
c    subtract multiples of it from rows i+1,i+2,...,ndim
c    to zero column i

         mask(i,:) = .false.
         where (mask)
            mattemp = mattemp - spread(mattemp(i,:),1,ndim) *
     +              spread(mattemp(:,i),2,2*ndim)
         endwhere
       enddo

       mat = mattemp(:,ndim+1:2*ndim)

       return
       end
```

B14 Tridiagonal Solver Routine

```
c     interpretation of Michael Mascagni's tridiagonal solver
c     originally written in C*

c     n can most likely be derived from dest.

      subroutine tridiag(dest,l,d,u,rhs)
      real dest,l,d,u,rhs
      integer offset,power
      integer address,lp,dp,up,rhsp,lm,dm,um,rhsm,alpha,gamma
      integer ll,ld,lu,lhs,temp,nn,array_context
      integer entry_vp_set,i
      include '/usr/include/cm/paris-configuration-fort.h'
      include '/usr/include/cm/CMF_defs.h'

      entry_vp_set = cm_current_vp_set()
      call cm_set_vp_set (cmf_get_vp_set_id(l))
      call cm_set_context()
      nn = cmf_get_axis_extent(l,0)-1

      address = cm_allocate_stack_field(32)
      ll = cmf_get_field_id(l)
      ld = cmf_get_field_id(d)
      lu = cmf_get_field_id(u)
      lrhs = cmf_get_field_id(rhs)
      alpha = cmf_get_field_id (dest)
      lp = cm_allocate_stack_field(32)
      dp = cm_allocate_stack_field (32)
      up = cm_allocate_stack_field(32)
      rhsp = cm_allocate_stack_field(32)
      lm = cm_allocate_stack_field (32)
      dm = cm_allocate_stack_field (32)
      um = cm_allocate_stack_field (32)
      rhsm = cm_allocate_stack_field (32)
      gamma = cm_allocate_stack_field (32)
      temp = cm_allocate_stack_field (1)
      array_context = cm_allocate_stack_field(1)

      call cm_my_news_coordinate_1l (address,0,32)
      call cm_u_eq_constant_1l(address,0,32)
      call cm_logand_context_with_test()
      do 2 i=1,cmf_get_rank(l)
         call cm_my_news_coordinate_1l (address,i,32)
         call cm_u_lt_constant_1l(address,cmf_get_axis_extent(l,i-1),32)
         call cm_logand_context_with_test()
```

```
2     continue
      call cm_store_context (array_context)
      call cm_my_news_coordinate_1L (addres- *,32)
      offset =1
      power =0
1     call cm_load_context (array_context)
      call cm_u_le_constant_1l (address,(nn-offset),32)
      call cm_store_test (temp)
      call cm_load_context(temp)
      call cm_get_from_power_two_1l(lp,ll,1,power,cm_upward,32)
      call cm_get_from_power_two_1l(dp,ld,1,powe-,cm_upward,32)
      call cm_get_from_power_two_1l(up,lu,1,power,cm_upward,32)
      call cm_get_from_power_two_1l(rhsp,lrhs,1,power,cm_upward
     !     ,32)
      call cm_f_multiply_constant_3_1l (gamma,lu,dble(-1.0),23,8)
      call cm_f_divide_2_1l (gamma,dp,23,8)
      call cm_load_context(array_context)
      call cm_logxor_constant_2_1l (temp,1,1)
      call cm_logand_context(temp)
      call cm_f_move_constant_1l (lp,dble(0.0),23,8)
      call cm_f_move_constant_1l (dp,dble(1.0),23,8)
      call cm_f_move_constant_1l (up,dble(0.0),23,8)
      call cm_f_move_constant_1l (rhsp,dble(0.0),23,8)
      call cm_f_move_constant_1l (gamma,dble(0.0),23,8)
      call cm_load_context (array_context)
      call cm_u_ge_constant_1l (address,offset,32)
      call cm_store_test (temp)
      call cm_logand_context_with_test()
      call cm_get_from_power_two_1l(lm,ll,1,power,cm_downward,
     !     32)
      call cm_get_from_power_two_1l(dm,ld,1,power,cm_downward,
     !     32)
      call cm_get_from_power_two_1l(um,lu,1,power,cm_downward,
     !     32)
      call cm_get_from_power_two_1l(rhsm,lrhs,1,power,
     !     cm_downward,32)
      call cm_f_multiply_constant_3_1l (alpha,ll,dble(-1.0),23,8)
      call cm_f_divide_2_1l (alpha,dm,23,8)
      call cm_load_context(array_context)
      call cm_logxor_constant_2_1l (temp,1,1)
      call cm_logand_context(temp)
      call cm_f_move_constant_1l (lm,dble(0.0),23,8)
      call cm_f_move_constant_1l (dm,dble(1.0),23,8)
      call cm_f_move_constant_1l (um,dble(0.0),23,8)
      call cm_f_move_constant_1l (rhsm,dble(0.0),23,8)
      call cm_f_move_constant_1l (alpha,dble(0.0),23,8)
      call cm_load_context(array_context)
      call cm_f_multiply_3_1l (ll,alpha,lm,23,8)
```

```
call cm_f_multiply_3_1l (lu,gamma,up,23,8)
call cm_f_mult_add_1l (ld,alpha,um,ld,23,8)
call cm_f_mult_add_1l (ld,gamma,lp,ld,23,8)
call cm_f_mult_add_1l (lrhs,alpha,rhsm,lrhs,23,8)
call cm_f_mult_add_1l (lrhs,gamma,rhsp,lrhs,23,8)
call cm_f_divide_3_1l (alpha,lrhs,ld,23,8)
power = power+1
offset = offset+offset
if (offset .le. nn) then
   goto 1
   endif
call cm_deallocate_stack_through (address)
call cm_set_vp_set (entry_vp_set)
call CMF_set_is_modified(dest,MODIF)
return
end
```