Chapter 5

Array Aliasing

A set of CM Fortran utility library procedures support *subgrid equivalencing*, a form of storage association for CM arrays. These procedures enable you to create an *array alias*, which addresses memory already allocated for another array but treats it as being of a different type, shape, or layout. No data motion occurs; the memory is simply associated with an additional array descriptor.

Some uses of array aliasing:

- You can take advantage of the compiler's optimizations on serial dimensions by creating a 2-dimensional **`(:SERIAL,:NEWS)`** alias for storage that is allocated as a 1-dimensional **`:NEWS`** array. The serial axis of the alias is storage-associated with the on-processor portion of the target array's **`:NEWS`** axis.
- You can write rank-independent subprograms. Pass a 1D actual array argument, but alias it in the subprogram as *n*D with suitable axis extents.
- You can manage memory manually by defining one huge 1D array and then use aliases to choose portions of it for specific tasks.
- You can reduce the compiler's use of temporaries in stencils, etc. Use two aliases into the same data rather than making a temporary copy of the data and shifting it.

This chapter provides a brief introduction to the procedures in the aliasing package. It also provides detailed descriptions of the individual procedures and explanations of the alias package's error codes. Three sample programs appear in Section 5.7.

Copyright © 1994 Thinking Machines Corporation

5.1 Restrictions and Caveats

- The aliasing procedures behave incorrectly if the **ARRAY** argument — the target of an alias constructor or the array queried by an inquiry procedure — is aligned in a strided or offset fashion with another array or aligned with the less rapidly varying axes of an array of higher rank.

- Beware of padding in either a target array or its alias. If the target array and the alias have different geometries, the internal garbage-masking software will mistake the location of garbage elements in the alias. The result can be that real data is masked and garbage data is computed upon and stored. Recall that the **cmf** switch **-nopadding** removes vector-length padding only; global padding, if any, is not affected by this switch.

- It is not advisable to pass to a subprogram two (or more) array arguments that address the same or overlapping memory if the subprogram changes either argument. The compiler assumes that argument memory does not overlap and conducts its dependence analysis and global optimizations (under the **-O** switch) on this assumption.

- An example of aliases confounding dependence analysis is seen in the compiler's internal strip-mining. For example, given vectors **A**, **B**, and **A1**, suppose that **A1** is aliased to the location offset by one element from the base of **A**. Then, consider a program executing under the CM-5 vector units model that contains the statements:

```
A = expression_1
B = A1 + expression_2
```

  If both these source lines are included in one local code block, the procedure is as follows: The code block turns into a loop where each iteration of the loop works on a vector of length 8. In the first iteration of the loop, elements 1:8 of **A** are updated. Note that these are elements 2:9 of **A1**. So the assignment to **B** is using new values for elements 1:7 of **A1**, and an old value for element 8 of **A1**.

Version 2.2, October 1994
Copyright © 1994 Thinking Machines Corporation

5.2 Constructing Aliases

Use these procedures to create array aliases:

```
CMF_ALIAS_TYPE( ALIAS, TYPE, ARRAY, RESIZE_MODE, ASTAT )

CMF_ALIAS_DETAILED( ALIAS, RANK, EXTENTS, ORDERS,
    SUBGRIDS, PMASKS, OFFSETS, MASK_CHECK, ARRAY, ASTAT )
```

The **ALIAS** argument is a front-end array that serves as an array descriptor; other arguments fill in the slots of the descriptor. The alias can be used as a CM array when it is passed to a subprogram that declares it as such.

The **ARRAY** argument can be either a CM array (including an alias in the callee scope) *or* a front-end array that is a valid array descriptor. If the latter, the argument could only be an alias or a descriptor generated by a **CMF_ALLOCATE_** routine, since these are the only arrays whose descriptors can be referenced directly.

The alias package has its own run-time safety mechanism. At the nondefault safety level, bounds-checking is performed to ensure that the alias addresses memory within the range addressed by the target **ARRAY**. At the default safety level, the alias may address any allocated memory.

Copyright © 1994 Thinking Machines Corporation

CMF_ALIAS_TYPE

Creates an array alias of a specified type.

Syntax

```
INCLUDE 'CMF_defs.h'

CALL CMF_ALIAS_TYPE
&    ( ALIAS, TYPE, ARRAY, RESIZE_MODE, ASTAT )
```

Arguments

ALIAS — An integer front-end array of size **CMF_SIZEOF_DESCRIPTOR** (a predefined constant).

TYPE — A predefined integer constant that indicates data type. It must be one of:

CMF_LOGICAL,

CMF_S_INTEGER, CMF_LONG_S_INTEGER

CMF_FLOAT, CMF_DOUBLE,

CMF_COMPLEX, CMF_DOUBLE_COMPLEX.

ARRAY — An allocated CM array of any type, including an array alias, *or* an integer front-end array that is a valid array descriptor.

RESIZE_MODE — A predefined integer constant that indicates which dimension of **ALIAS**, if any, to adjust to accommodate a data type different from **ARRAY**. See below for explanation. It must be one of:

CMF_MUTATE_SELECT

CMF_MUTATE_ANY

CMF_MUTATE_BEST

ASTAT — An integer scalar into which the error status of the operation is placed. Zero indicates success; a positive value indicates failure.

If the **RESIZE_MODE** argument is **CMF_MUTATE_SELECT**, **ASTAT** is used to indicate which alias dimension (beginning from axis 0, not 1) to resize, as explained below. **ASTAT** retains its conventional return semantics.

Version 2.2, October 1994
Copyright © 1994 Thinking Machines Corporation

Result

None.

Description

Fills in the **ALIAS** argument so that it may be passed to a subprogram that declares it as a CM array. The alias addresses the same data addressed by the argument **ARRAY**, but treats it as the data type represented by **TYPE**. If the new type is not the same element size as the type of **ARRAY**, the **RESIZE_MODE** argument indicates how to adjust the size of the array represented by **ALIAS** with respect to the argument **ARRAY**.

Description of Resizing Modes

The **RESIZE_MODE** argument is ignored if the element size of the **TYPE** argument is the same as element size of the argument **ARRAY**. If this is not the case, then the length of one of the dimensions of **ALIAS** will differ from the length of the corresponding dimension of **ARRAY**. The **RESIZE_MODE** argument controls which dimension is selected for adjustment.

- **CMF_MUTATE_SELECT** implies that the **ASTAT** argument has been initialized to the axis number that is to be adjusted.
- **CMF_MUTATE_ANY** allows the implementation to select which axis is to be adjusted. In the current implementation, the outermost axis (the one with largest local, or per-processor, stride) is selected if the size of the new data type is less than the size of the old data type. If the size of the new data type is greater than the size of the old data type, preference is given to the outermost axis, if any, of which the length in the original array was a multiple of the ratio of data type size expansion. If, in this case, no axis is a multiple of the expansion ratio, the locally outermost axis will be extended as necessary to ensure that the generated alias accesses at least all of the memory accessible through the original array. Attempts to operate on the outermost elements are not guaranteed success under these circumstances.
- **CMF_MUTATE_BEST** is essentially the same as **CMF_MUTATE_ANY**, unless the data type expansion ratio is greater than one and no dimension of the original array bears a local extent that is a multiple of that ratio, in which case an error is returned. In this mode, all elements of the generated alias array can be accessed successfully.

Copyright © 1994 Thinking Machines Corporation

Restrictions

- The type specified by **CMF_LONG_S_INTEGER** is supported only on CM-5 systems with vector units.
- The aliasing procedures behave incorrectly if the **ARRAY** argument is aligned in a strided or offset fashion with another array or aligned with the less rapidly varying axes of an array of higher rank.

Version 2.2, October 1994
Copyright © 1994 Thinking Machines Corporation

CMF_ALIAS_DETAILED

Creates an array alias with a specified shape and detailed layout.

Syntax

```
 INCLUDE 'CMF_defs.h'

 CALL CMF_ALIAS_DETAILED
&    ( ALIAS, RANK, EXTENTS, ORDERS, SUBGRIDS, PMASKS,
&      OFFSETS, MASK_CHECK, ARRAY, ASTAT )
```

Arguments

ALIAS	An integer front-end array of size **CMF_SIZEOF_DESCRIPTOR** (a predefined constant). This argument becomes the alias descriptor.
RANK	An integer scalar indicating the rank of the alias.
EXTENTS	An integer front-end vector that is at least **RANK** in length and contains the dimension extents of the alias.
ORDERS	An integer front-end vector that is at least **RANK** in length and contains values specifying axis ordering of the alias. Each value must be one of: CMF_SERIAL_ORDER CMF_NEWS_ORDER
SUBGRIDS	An integer front-end vector that is at least **RANK** in length and contains the subgrid length for each axis of the alias.
PMASKS	An integer front-end vector that is at least **RANK** in length and contains bit-masks to specify processors. If the **ORDERS** argument contains the value **CMF_SERIAL_ORDER** for any axis, the **PMASKS** argument must contain zero for that axis.

Copyright © 1994 Thinking Machines Corporation

OFFSETS — An integer front-end vector that is at least the rank of the target array **ARRAY** in length and contains a subgrid-relative zero-based offset index for each dimension of **ARRAY**. The address of the element of the argument **ARRAY**'s local subgrid that is selected by the values in **OFFSETS** is used as the base address of the array represented by **ALIAS**. The values that correspond to purely inter-processor dimensions (those whose local extent is 1) must be zero.

If all offsets are zero, **%VAL(0)** may be passed instead of an **OFFSETS** array of the proper length containing all zeros.

MASK_CHECK — A logical scalar that specifies whether to perform a safety check to ensure that the same garbage mask, if any, is used by the geometries of both the **ARRAY** argument and the generated alias. The error status of the operation is returned in **ASTAT**.

ARRAY — An allocated CM array of any type, including an array alias, *or* an integer front-end array that is a valid array descriptor.

ASTAT — An integer scalar into which the error status of the operation is placed. Zero indicates success; a positive value indicates failure.

If alias safety is nonzero and **OFFSETS** address memory out of the bounds of the argument **ARRAY**, a nonzero value is returned in **ASTAT**.

Result

None.

Description

Fills in the **ALIAS** argument with a detailed array layout so that it may be passed to a subprogram that declares it as a CM array. The alias treats the memory it addresses as having the shape described by **RANK** and **EXTENTS** and the layout described by **ORDERS**, **SUBGRIDS**, and **PMASKS**. The alias addresses memory beginning at the location computed from the base location of the argument **ARRAY**, as offset by the subgrid-relative values in **OFFSETS**.

Version 2.2, October 1994
Copyright © 1994 Thinking Machines Corporation

Restrictions

- The aliasing procedures behave incorrectly if the **ARRAY** argument is aligned in a strided or offset fashion with another array or aligned with the less rapidly varying axes of an array of higher rank.

See Also

CMF_GET_AXIS_ORDERS
CMF_GET_LOCAL_EXTENTS
CMF_GET_LOCAL_STRIDES
CMF_GET_PROCESSOR_MASKS

Copyright © 1994 Thinking Machines Corporation

5.3 Operating upon Aliases

Use these procedures to identify, mutate, or remove an alias.

```
LOG = CMF_ARRAY_IS_ALIAS( ARRAY, ASTAT )

CMF_SET_ALIAS_LOCATION( ALIAS, LOCATION, ASTAT )

CMF_REMOVE_ALIAS( ALIAS, ASTAT )
```

These procedures may be applied to aliases in either declaration scope — that is, either as front-end arrays in the routine that constructs an alias or as CM arrays in a subprogram. The second two procedures should be applied *only* to array aliases; at the default safety level, their application to non-aliases is not detected and may lead to incorrect results.

Version 2.2, October 1994
Copyright © 1994 Thinking Machines Corporation

CMF_ARRAY_IS_ALIAS

Reports whether the argument **ARRAY** is a generated array alias.

Syntax

```
INCLUDE 'CMF_defs.h'

LOGICAL = CMF_ARRAY_IS_ALIAS( ARRAY, ASTAT )
```

Arguments

ARRAY	An allocated CM array of any type, including an array alias, *or* an integer front-end array that is a valid array descriptor.
ASTAT	An integer scalar into which the error status of the operation is placed. Zero indicates success; a positive value indicates that the **ARRAY** argument is not a valid CM array or array alias.

Result

Logical scalar. Result is **.TRUE.** if the argument **ARRAY** is a generated array alias, **.FALSE.** otherwise.

See Also

CMF_ALIAS_TYPE
CMF_ALIAS_DETAILED

Copyright © 1994 Thinking Machines Corporation

CMF_SET_ALIAS_LOCATION

Modifies an alias so that the base location of the CM array it represents is the parallel memory address in **LOCATION**.

Syntax

```
INCLUDE 'CMF_defs.h'

CALL CMF_SET_ALIAS_LOCATION( ALIAS, LOCATION, ASTAT )
```

Arguments

ALIAS	An integer front-end array that is the descriptor of a generated array alias.
LOCATION	An integer scalar that contains a parallel memory address. The value is that returned by **CMF_GET_GLOBAL_LOCATION** or **CMF_GET_PHYSICAL_LOCATION**.
ASTAT	An integer scalar into which the error status of the operation is placed. Zero indicates success; a positive value indicates failure. If alias safety is nonzero and **ALIAS** is not a generated array alias, a nonzero value is returned in **ASTAT**.

Result

None.

See Also

CMF_GET_GLOBAL_LOCATION
CMF_GET_PHYSICAL_LOCATION

Version 2.2, October 1994
Copyright © 1994 Thinking Machines Corporation

CMF_REMOVE_ALIAS

Deallocates the front-end auxiliary and substructure memory associated with an alias descriptor.

Syntax

```
INCLUDE 'CMF_defs.h'

CALL CMF_REMOVE_ALIAS( ALIAS, ASTAT )
```

Arguments

ALIAS	An integer front-end array that is the descriptor of a generated array alias. (Any attempt to use the descriptor as an alias after removal is an error.)
ASTAT	An integer scalar into which the error status of the operation is placed. Zero indicates success; a positive value indicates failure. If alias safety is non-zero and **ALIAS** is not a generated array alias, a nonzero value is returned in **ASTAT** and no deallocation takes place.

Result

None.

See Also

CMF_ALIAS_TYPE
CMF_ALIAS_DETAILED

Copyright © 1994 Thinking Machines Corporation

5.4 Array Descriptor Inquiries

Use these procedures to derive information about an existing array or alias, usually for the purpose of constructing arguments to an alias constructor or mutator.

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

INT = CMF_GET_PHYSICAL_LOCATION( ARRAY, OFFSETS, ASTAT )

CMF_GET_AXIS_ORDERS( ARRAY, ORDERS, RANK, ASTAT )

CMF_GET_LOCAL_EXTENTS( ARRAY, EXTENTS, RANK, ASTAT )

CMF_GET_LOCAL_STRIDES( ARRAY, STRIDES, RANK, ASTAT )

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


Version 2.2, October 1994
Copyright © 1994 Thinking Machines Corporation

CMF_GET_GLOBAL_LOCATION

Returns as an integer the location (a parallel memory address) of the element of the argument **ARRAY** selected by the argument **OFFSETS**.

Syntax

```
INCLUDE 'CMF_defs.h'

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

Arguments

ARRAY	An allocated CM array of any type, including an array alias, *or* an integer front-end array that is a valid array descriptor.
OFFSETS	An integer front-end vector that is at least **RANK(ARRAY)** in length and that contains an index offset for each dimension of **ARRAY**. The offset values index **ARRAY** as though all dimensions were declared as zero-based.
ASTAT	An integer scalar into which the error status of the operation is placed. Zero indicates success; a positive value indicates failure. If alias safety is nonzero and **OFFSETS** address memory out of the bounds of the argument **ARRAY**, a nonzero value is returned in **ASTAT**.

Result

Integer scalar representing a parallel memory address.

Restriction

- The aliasing procedures behave incorrectly if the **ARRAY** argument is aligned in a strided or offset fashion with another array or aligned with the less rapidly varying axes of an array of higher rank.

Copyright © 1994 Thinking Machines Corporation

See Also

CMF_SET_ALIAS_LOCATION

Version 2.2, October 1994
Copyright © 1994 Thinking Machines Corporation

CMF_GET_PHYSICAL_LOCATION

Returns as an integer the location (a parallel memory address) of the element of the argument **ARRAY**'s local subgrid selected by the argument **OFFSETS**.

Syntax

```
INCLUDE 'CMF_defs.h'

INTEGER = CMF_GET_PHYSICAL_LOCATION( ARRAY, OFFSETS, ASTAT )
```

Arguments

ARRAY	An allocated CM array of any type, including an array alias, *or* an integer front-end array that is a valid array descriptor.
OFFSETS	An integer front-end vector that is at least **RANK(ARRAY)** in length and contains a subgrid-relative index offset for each dimension of **ARRAY**. The offsets are zero-based indices addressing the in-processor subgrid of the global array. The values that correspond to purely inter-processor dimensions (those whose local extent is 1) must be zero.
ASTAT	An integer scalar into which the error status of the operation is placed. Zero indicates success; a positive value indicates failure. If alias safety is nonzero and **OFFSETS** address memory out of the bounds of the argument **ARRAY**, a nonzero value is returned in **ASTAT**.

Result

Integer scalar representing a parallel memory address.

Restriction

- The aliasing procedures behave incorrectly if the **ARRAY** argument is aligned in a strided or offset fashion with another array or aligned with the less rapidly varying axes of an array of higher rank.

Copyright © 1994 Thinking Machines Corporation

See Also

CMF_SET_ALIAS_LOCATION

Version 2.2, October 1994
Copyright © 1994 Thinking Machines Corporation

CMF_GET_AXIS_ORDERS

Fills the **ORDERS** argument array with values indicating the ordering (serial, NEWS, or send) of each axis of the argument **ARRAY**.

Syntax

```
INCLUDE 'CMF_defs.h'

CALL CMF_GET_AXIS_ORDERS( ARRAY, ORDERS, RANK, ASTAT )
```

Arguments

ARRAY	An allocated CM array of any type, including an array alias, *or* an integer front-end array that is a valid array descriptor.
ORDERS	An integer front-end vector that is at least **RANK(ARRAY)** in length. The values returned in **ORDERS** are one or more of: CMF_SERIAL_ORDER CMF_NEWS_ORDER
RANK	An integer scalar. The array rank is returned in **RANK**.
ASTAT	An integer scalar into which the error status of the operation is placed. Zero indicates success; a positive value indicates failure.

Result

None.

Restrictions

- This procedure returns **CMF_SERIAL_ORDER** only for axes that are explicitly declared **:SERIAL** or **CMF_SERIAL_ORDER**, not for local axes that are declared with **:PROCS** or **:PDESC** syntax.
- This procedure is incompatible with the **cmf** option **-noaxisreorder**. When compiled with this option, the procedure returns **CMF_NEWS_ORDER** for all axes.

Copyright © 1994 Thinking Machines Corporation

- The aliasing procedures behave incorrectly if the **ARRAY** argument is aligned in a strided or offset fashion with another array or aligned with the less rapidly varying axes of an array of higher rank.

See Also

CMF_ALIAS_DETAILED

Version 2.2, October 1994
Copyright © 1994 Thinking Machines Corporation

CMF_GET_LOCAL_EXTENTS

Fills the **EXTENTS** argument array with the axis extents of **ARRAY**'s per-processor subgrid, and the **RANK** argument with the rank of **ARRAY**'s per-processor subgrid.

Syntax

```
INCLUDE 'CMF_defs.h'

CALL CMF_GET_LOCAL_EXTENTS( ARRAY, EXTENTS, RANK, ASTAT )
```

Arguments

ARRAY	An allocated CM array of any type, including an array alias, *or* an integer front-end array that is a valid array descriptor.
EXTENTS	An integer front-end vector that is at least **RANK(ARRAY)** in length. The subgrid extents (excluding padding, if any) are returned in **EXTENTS**.
RANK	An integer scalar. The subgrid rank is returned in **RANK**.
ASTAT	An integer scalar into which the error status of the operation is placed. Zero indicates success; a positive value indicates failure.

Result

None.

Restriction

- The aliasing procedures behave incorrectly if the **ARRAY** argument is aligned in a strided or offset fashion with another array or aligned with the less rapidly varying axes of an array of higher rank.

Copyright © 1994 Thinking Machines Corporation

See Also

CMF_ALIAS_DETAILED

Version 2.2, October 1994
Copyright © 1994 Thinking Machines Corporation

CMF_GET_LOCAL_STRIDES

Fills the **STRIDES** argument array with the elemental strides of the axes of **ARRAY**'s per-processor subgrid, and the **RANK** argument with the rank of **ARRAY**'s per-processor subgrid.

Syntax

```
INCLUDE 'CMF_defs.h'

CALL CMF_GET_LOCAL_STRIDES( ARRAY, STRIDES, RANK, ASTAT )
```

Arguments

ARRAY	An allocated CM array of any type, including an array alias, *or* an integer front-end array that is a valid array descriptor.
STRIDES	An integer front-end vector that is at least **RANK(ARRAY)** in length. The subgrid elemental axis strides are returned in **STRIDES**.
RANK	An integer scalar. The subgrid rank is returned in **RANK**.
ASTAT	An integer scalar into which the error status of the operation is placed. Zero indicates success; a positive value indicates failure.

Result

None.

Description

The returned values should be interpreted such that: the innermost subgrid-local axis has a unit stride, while the outermost subgrid-local axis has a stride equal to the product of the strides of all inferior axes, multiplied by the local (per-processor) extent of the axis with the next smaller stride.

Copyright © 1994 Thinking Machines Corporation

Restriction

- The aliasing procedures behave incorrectly if the **ARRAY** argument is aligned in a strided or offset fashion with another array or aligned with the less rapidly varying axes of an array of higher rank.

See Also

CMF_ALIAS_DETAILED

Version 2.2, October 1994
Copyright © 1994 Thinking Machines Corporation

CMF_GET_PROCESSOR_MASKS

Fills the **MASKS** argument array with integer bitmasks that describe the layout of **ARRAY**'s axes across processors, and fills the **RANK** argument with **ARRAY**'s rank.

Syntax

```
INCLUDE 'CMF_defs.h'

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

Arguments

ARRAY	An allocated CM array of any type, including an array alias, *or* an integer front-end array that is a valid array descriptor.
MASKS	An integer front-end vector that is at least **RANK(ARRAY)** in length. Each element is filled with an integer bitmask such that the intersection of any two elements is zero and the union of all masks is less than or equal to the value of **CMF_NUMBER_OF_PROCESSORS()-1**. Each bit of a given element is set if and only if the corresponding bit of the physical processor address is allocated to the axis of the argument **ARRAY** which corresponds to that element of **MASKS**.
RANK	An integer scalar. The rank of **ARRAY** is returned in **RANK**.
ASTAT	An integer scalar into which the error status of the operation is placed. Zero indicates success; a positive value indicates failure.

Result

None.

Restriction

- The aliasing procedures behave incorrectly if the **ARRAY** argument is aligned in a strided or offset fashion with another array or aligned with the less rapidly varying axes of an array of higher rank.

Copyright © 1994 Thinking Machines Corporation

See Also

CMF_ALIAS_DETAILED

Version 2.2, October 1994
Copyright © 1994 Thinking Machines Corporation

5.5 Execution Environment

Use these procedures to inquire about or set a feature of the execution environment for the aliasing procedures.

```
INT = CMF_ARRAY_SIZE_QUANTUM( )

CMF_SET_ALIAS_SAFETY_LEVEL( SAFETY_LEVEL )

INT = CMF_GET_ALIAS_SAFETY_LEVEL( )
```

The run-time checks performed at the non-default safety level are listed below in the reference descriptions of the two safety-related procedures. The error codes that may be returned in the **ASTAT** argument are described in Section 5.6.

Copyright © 1994 Thinking Machines Corporation

CMF_ARRAY_SIZE_QUANTUM

Returns the integer quantum by which virtual grid size increases in the current execution environment.

Syntax

```
INCLUDE 'CMF_defs.h'

INTEGER = CMF_ARRAY_SIZE_QUANTUM( )
```

Arguments

None.

Result

Integer scalar:

- Under CM-5 vector-units model, returns 1 with **-nopadding**, or 8 with the default **-padding**. The vector register in the CM-5 vector units is of length 8.
- Under CM-5 SPARC nodes model, returns 1.

Description

The result of this function indicates the vector-length, if any, of which the non-serial subgrid of a virtual grid must be a multiple. This factor, when multiplied by the current number of physical processors, gives the minimum number of elements a user array's non-serial dimensions must have to be stored without padding.

Version 2.2, October 1994
Copyright © 1994 Thinking Machines Corporation

CMF_SET_ALIAS_SAFETY_LEVEL

Sets level of run-time alias safety checking.

Syntax

```
INCLUDE 'CMF_defs.h'

CALL CMF_SET_ALIAS_SAFETY_LEVEL( SAFETY_LEVEL )
```

Arguments

SAFETY_LEVEL	Integer scalar. The value zero indicates the default safety level; any positive value indicates extra safety checking. Negative values are not supported. The former safety level is returned in the argument.

Result

None.

Description

Run-time alias safety checking may be set to perform either the default checks only or extra checks. Error status of an alias procedure is returned in the argument **ASTAT**.

At the higher safety level, the system checks for:

- attempts to construct an alias that addresses memory outside the range of memory addressed by the alias target
- attempts to dereference pointer elements of the descriptors passed to an alias routine (which is likely to result in a segmentation violation, if the descriptor argument is invalid or erroneous in any way)
- axis descriptor masks of a form inappropriate to the current execution environment

Copyright © 1994 Thinking Machines Corporation

See Also

CMF_GET_ALIAS_SAFETY_LEVEL

Version 2.2, October 1994
Copyright © 1994 Thinking Machines Corporation

CMF_GET_ALIAS_SAFETY_LEVEL

Returns the current level of run-time alias safety checking.

Syntax

```
INCLUDE 'CMF_defs.h'

INTEGER = CMF_GET_ALIAS_SAFETY_LEVEL( )
```

Arguments

None.

Result

Integer scalar. Zero indicates the default safety level; nonzero indicates extra safety checking.

Description

Run-time alias safety checking may be set to perform either the default checks only or extra checks. Error status of an alias procedure is returned in the argument **ASTAT**.

At the higher safety level, the system checks for:

- attempts to construct an alias that addresses memory outside the range of memory addressed by the alias target
- attempts to dereference pointer elements of the descriptors passed to an alias routine (which is likely to result in a segmentation violation, if the descriptor argument is invalid or erroneous in any way)
- axis descriptor masks of a form inappropriate to the current execution environment

Copyright © 1994 Thinking Machines Corporation

See Also

CMF_SET_ALIAS_SAFETY_LEVEL

Version 2.2, October 1994
Copyright © 1994 Thinking Machines Corporation

5.6 Alias Safety Checking: Error Codes

This section describes the integer error codes that are returned in the **ASTAT** argument when an aliasing operation fails. Most reflect default-level safety checking in the alias package.

If the environment is set to a higher level of safety checking (with a positive argument to **CMF_SET_ALIAS_SAFETY_LEVEL**), the system checks for the following conditions. If necessary, it returns in **ASTAT** one of the error codes identified as "with bounds checking enabled."

- attempts to construct an alias that addresses memory outside the range of memory addressed by the alias target
- attempts to dereference pointer elements of the descriptors passed to an alias routine (which is likely to result in a segmentation violation, if the descriptor argument is invalid or erroneous in any way)
- axis descriptor masks of a form inappropriate to the current execution environment

The error codes are defined in the header file **CMF_defs.h**. The code identified as "with mask checking enabled" refers to the argument **MASK_CHECK** to **CMF_ALIAS_DETAILED**.

CMF_error_null_array
: This code is returned if an attempt is made to alias an array with a null extent.

CMF_error_sparse_mask
: This code is returned if an attempt is made to create an alias with a discontiguous processor allocation mask along some axis, when the current execution environment does not support such a mask.

CMF_error_axis_order
: This code is returned if an attempt is made to alias a descriptor with an invalid or otherwise unrecognized axis ordering along some axis, or to generate an alias with an invalid or otherwise unrecognized axis ordering along some axis.

Copyright © 1994 Thinking Machines Corporation

CMF_error_cannot_intern

This code is returned if an attempt to intern a run-time system geometry when generating an alias fails for any reason. No alias is generated in this case.

CMF_error_bad_argument

This code is returned if one of the arguments to an alias support routine is out of range or otherwise invalid and no more specific error code applies.

CMF_error_bad_geometry

This code is returned if the alias target CM array descriptor (by convention, the argument prototyped with the name **ARRAY**) is determined to be corrupt in its geometry components.

CMF_error_mask_mismatch

With mask checking enabled, this code is returned if an attempt is made to generate an alias such that its geometry involves a garbage mask which does not match the garbage mask of the alias target (by convention, the argument prototyped with the name **ARRAY**).

CMF_error_fe_home

This code is returned by the **CMF_ALIAS_DETAILED** subroutine if its **ARRAY** argument is a front-end array that is not a valid array descriptor.

CMF_error_fe_memory

This code is returned if an attempt to allocate heap memory for alias descriptor substructures on the control processor fails.

CMF_error_bad_type

This code is returned if an attempt is made to generate a type alias (as with **CMF_ALIAS_TYPE**) of an unsupported type. Currently supported type constants are:

```
CMF_LOGICAL
CMF_S_INTEGER
CMF_LONG_S_INTEGER
CMF_FLOAT
CMF_DOUBLE
CMF_COMPLEX
CMF_DOUBLE_COMPLEX
```


Version 2.2, October 1994
Copyright © 1994 Thinking Machines Corporation

CMF_error_bad_opcode
: This code is returned if an attempt is made to generate an alias using an undefined or unsupported **RESIZE_MODE** argument to **CMF_ALIAS_TYPE**.

CMF_error_bounds_violation
: With bounds-checking enabled, this code is returned if an attempt is made to construct an alias referencing CM memory that was not referenced by the original alias target (by convention, the argument prototyped with the name **ARRAY**).

CMF_error_no_axis
: This code is returned if an attempt is made to alias a target array in such a fashion that a run-time geometry axis of the target array that is not used in the user layout of that array is indexed by an **OFFSETS** argument of the alias routine.

CMF_error_no_alias
: This code is returned if an attempt is made to remove an alias, and the argument alias provided is not recognized as a generated alias.

CMF_error_aligned_geometry
: This code is returned if an attempt is made to generate an alias for a target array that is detectably aligned to another CM array.

CMF_error_runtime_layer
: This code is returned if the current execution environment does not support the alias operation being attempted, or if the user code was linked with a run-time system not appropriate to the architecture supported by the linked utility library.

Copyright © 1994 Thinking Machines Corporation

5.7 Examples of Detailed-Alias Construction

Example 1

This program aliases a 1D NEWS-ordered array to a 2D **:SERIAL**, **:NEWS** array. The alias treats the local (on-processor) portion of the target **:NEWS** array as a separate **:SERIAL** axis. The program output shown afterward gives the layout and contents of the target array and the alias.

```
      PROGRAM SIMPLE_ALIASING
C     Get size of 1D array to use in aliasing program
      INTEGER N

      PRINT *,'Simple array aliasing program'
      PRINT *,'What 1D array size do you want? '
      READ  *,N
      CALL MAIN(N)
      END PROGRAM SIMPLE_ALIASING

      SUBROUTINE MAIN(N)
C     Define array and alias, then call printing subroutine

      INCLUDE 'CMF_defs.h'
      INTEGER N,A(N)
CMF$  LAYOUT A(:NEWS)
      INTEGER SUBGRID_SIZES(1), OLD_PMASKS(1)
      INTEGER A_ALIAS(CMF_SIZEOF_DESCRIPTOR)
      INTEGER EXTENTS(2), SGLENGTHS(2), ORDERS(2)
      INTEGER PMASKS(2), OFFSETS(1)
      LOGICAL MASK_CHECK
      INTEGER PROCS,RANK
      INTEGER I,J,STATUS

C Zero the real (target)array and then self-index its elements
      A=0
      FORALL (I=1:N) A(I) = I

C Get information about subgrid size and pmasks of array
      PROCS = CMF_NUMBER_OF_PROCESSORS()
      CALL CMF_GET_LOCAL_EXTENTS(A,SUBGRID_SIZES,RANK,STATUS)
      CALL CMF_GET_PROCESSOR_MASKS(A,OLD_PMASKS,RANK,STATUS)
```


Version 2.2, October 1994
Copyright © 1994 Thinking Machines Corporation

```
C Define properties of the alias array:
      RANK = 2

C Axis 1 of alias is SERIAL order axis, with same subgrid
C size as in the original array
      ORDERS(1)=CMF_SERIAL_ORDER
      EXTENTS(1)=SUBGRID_SIZES(1)
      SGLENGTHS(1)=SUBGRID_SIZES(1)
      PMASKS(1)=0

C Axis 2 of alias is NEWS order axis, with extent equal to
C the number of CM processors executing the program
      ORDERS(2)=CMF_NEWS_ORDER
      EXTENTS(2)=PROCS
      SGLENGTHS(2)=1
      PMASKS(2)=PROCS-1

C Don't bother checking the alias's garbage mask
      MASK_CHECK=.FALSE.
      OFFSETS(1)=0

C Define the alias array
      CALL CMF_ALIAS_DETAILED
     $    (A_ALIAS,RANK,EXTENTS,ORDERS,SGLENGTHS,
     $    PMASKS,OFFSETS,MASK_CHECK,A,STATUS)

C Print out the contents of both arrays
      CALL DISPLAY_ARRAYS(A,N,A_ALIAS,EXTENTS(1),EXTENTS(2))
      END SUBROUTINE MAIN

      SUBROUTINE DISPLAY_ARRAYS(ARRAY,SIZE,ALIAS,SUBGRID,PROCS)
C Print contents and descriptions of target array and alias

      INCLUDE 'CMF_defs.h'
      INTEGER SIZE,SUBGRID,PROCS
      INTEGER ARRAY(SIZE), ALIAS(SUBGRID,PROCS)
CMF$  LAYOUT ARRAY(:NEWS), ALIAS(:SERIAL,:NEWS)
      INTEGER I,J
      PRINT *,' '
      PRINT *,'Here''s the original 1D array:'
      PRINT *,' '
      CALL CMF_DESCRIBE_ARRAY(ARRAY)
      PRINT *,' '
      PRINT '(1000I4)', (ARRAY(I), I=1,SIZE)
      PRINT *,' '
```


Copyright © 1994 Thinking Machines Corporation

```
      PRINT *,'And here''s the 2D alias array:'
      PRINT *,' '
      CALL CMF_DESCRIBE_ARRAY(ALIAS)
      PRINT *,' '
      DO I=1,SUBGRID
         PRINT '(1000I4)', (ALIAS(I,J), J=1,PROCS)
      END DO
      END SUBROUTINE DISPLAY_ARRAYS
```

Example 1 produces the following output when executed on a 16-processor CM-5: a 4-node, 16-vector-unit partition under the CM Fortran switches **-vu** and **-padding**.

```
% a.out
Simple array aliasing program
What 1D array size do you want?
116

Here's the original 1-D array:
    ...
    user_rank                :  1
    axes_extents             :  116
    …
Machine geometry id: 0x147500, rank: 1, column major
    Machine geometry elements:    128
    Overall subgrid size:         8
    Axis 0:
         Extent:        128 (16 physical x 8 subgrid)
         Off-chip:      4 bits, mask = 0xf
         Subgrid:       length = 8, axis-increment = 1

 1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16
17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32
33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48
49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64
65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80
81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96
97  98  99 100 101 102 103 104 105 106 107 108 109 110 111 112
113 114 115 116
```


Copyright © 1994 Thinking Machines Corporation

```
And here's the 2-D alias array:
    ...
    user_rank                :  2
    axes_extents             :  8 16
    ...
Machine geometry id: 0x147780, rank: 2, column major
    Machine geometry elements:    128
    Overall subgrid size:         8
    Axis 0 (serial axis):
         Extent:        8 (1 physical x 8 subgrid)
         Off-chip:      0 bits, mask = 0x0
         Subgrid:       length = 8, axis-increment = 1
    Axis 1:
         Extent:        16 (16 physical x 1 subgrid)
         Off-chip:      4 bits, mask = 0xf
         Subgrid:       length = 1, axis-increment = 1

  1   9  17  25  33  41  49  57  65  73  81  89  97 105 113   0
  2  10  18  26  34  42  50  58  66  74  82  90  98 106 114   0
  3  11  19  27  35  43  51  59  67  75  83  91  99 107 115   0
  4  12  20  28  36  44  52  60  68  76  84  92 100 108 116   0
  5  13  21  29  37  45  53  61  69  77  85  93 101 109   0   0
  6  14  22  30  38  46  54  62  70  78  86  94 102 110   0   0
  7  15  23  31  39  47  55  63  71  79  87  95 103 111   0   0
  8  16  24  32  40  48  56  64  72  80  88  96 104 112   0   0
```


Copyright © 1994 Thinking Machines Corporation

Example 2

This program illustrates array aliasing in a Laplace solver that uses successive relaxation. Array aliasing is useful because the program's performance benefits from using two different array addressing schemes to address the same memory locations.

Such a solver is easily implemented in CM Fortran by shifting all data elements to get nearest-neighbor values. However, a more efficient implementation is to move only off-processor values (to reduce excessive on-processor data movement). Instead of viewing the data arrays as 2D, then, this approach views the same data as 4D: The first two axes address elements within a subgrid and the last two axes address processing-element number.

NOTE: This program is incompatible with vector-length padding of subgrids. Thus, if you use the CM-5 vector-units execution model, you must compile with the **`cmf`** switch **`-nopadding`**. Use of the switch **`-noaxis`** is optional but may affect performance. Recall that the target array should be allocated without global padding as well; specify input values whose product is an integer multiple of partition size.

The Fortran **`D`** lines are used for optional timing calls.

```
      PROGRAM ALIAS_STENCIL
      DO
         WRITE(6,*)'NX,NY,NLOOP (Enter "0 0 0" to exit) ?'
         READ(5,*)NX,NY,NLOOP
         IF (NX.LE.0) EXIT
         CALL SUB(NX,NY,NLOOP)
      END DO

      STOP
      END

      SUBROUTINE SUB(NX,NY,NLOOP)

      INCLUDE 'CMF_defs.h'
D     INCLUDE 'timer-fort.h'

      INTEGER, ARRAY(CMF_SIZEOF_DESCRIPTOR) :: ALIASX, ALIASC
      INTEGER EXTENTS(4)
      DOUBLE PRECISION, DIMENSION(NX,NY) :: X, Y, COEF
CMF$  LAYOUT ALIASX(:SERIAL), ALIASC(:SERIAL), EXTENTS(:SERIAL)
CMF$  LAYOUT X(,), Y(,), COEF(,)
```


Version 2.2, October 1994
Copyright © 1994 Thinking Machines Corporation

```
C Initialize the data
      X=0.d0
      X(1,:)=1.d0
      X(NX,:)=1.d0

C Make a copy of the data
      Y=X

C Set relaxation coefficient to 0.1d0 in interior
C and 0.d0 otherwise
      COEF=0.d0
      COEF(2:NX-1,2:NY-1)=0.1d0

C CSHIFT method
D     CALL CM_timer_clear(0)
D     CALL CM_timer_start(0)
      DO (NLOOP) TIMES
         Y=Y+COEF*(
     .      0.25d0*(
     .         CSHIFT(Y,DIM=1,SHIFT=-1)
     .         +CSHIFT(Y,DIM=1,SHIFT=+1)
     .         +CSHIFT(Y,DIM=2,SHIFT=-1)
     .         +CSHIFT(Y,DIM=2,SHIFT=+1)
     .      )
     .      -Y
     .      )
      END DO
D     CALL CM_timer_stop(0)
D     WRITE(6,'(f10.5," seconds for CSHIFT method")')
D    .   CM_timer_read_elapsed(0)

C Alias method

C Set up array aliases
      CALL MAKE_ALIAS(X,ALIASX,EXTENTS)
      CALL MAKE_ALIAS(COEF,ALIASC,EXTENTS)

C Call routine
D     CALL CM_timer_clear(0)
D     CALL CM_timer_start(0)
      CALL STENCIL(NLOOP,ALIASX,ALIASC,
     .   EXTENTS(1),EXTENTS(2),EXTENTS(3),EXTENTS(4))
D     CALL CM_timer_stoP(0)
D     WRITE(6,'(f10.5," seconds for alias method")')
D    .   CM_timer_read_elapsed(0)
```


Copyright © 1994 Thinking Machines Corporation

```
C Remove aliases
      CALL CMF_REMOVE_ALIAS(ALIASC,IERROR)
      CALL CMF_REMOVE_ALIAS(ALIASX,IERROR)

C Report results
      WRITE(6,*)'maximum error: ',MAXVAL(ABS(X-Y))

      RETURN
      END



      SUBROUTINE MAKE_ALIAS(X,ALIAS,EXTENTS)

C This routine assumes X describes an array with only :NEWS
C axes. Here, the array should be at most 3D. The constructed
C alias addresses the same memory with twice as many axes:
C -- one :SERIAL and one :NEWS axis for each of the original
C axes.  The :SERIAL axes are all to the left. The :NEWS axes
C are all off-processor.
C The axis extents of the alias are returned in EXTENTS.

      INCLUDE 'CMF_defs.h'

      INTEGER, DIMENSION(CMF_SIZEOF_DESCRIPTOR) :: X, ALIAS
      INTEGER, DIMENSION(6)::EXTENTS,ORDERS,SUBGRID,PMASKS,
     &                       OFFSETS
CMF$  LAYOUT X(:SERIAL), ALIAS(:SERIAL)
CMF$  LAYOUT EXTENTS(:SERIAL), ORDERS(:SERIAL)
CMF$  LAYOUT SUBGRIDS(:SERIAL), PMASKS(:SERIAL),
CMF      OFFSETS(:SERIAL)

C Form subgrid extents (and get the number of dimensions)
      CALL CMF_GET_LOCAL_EXTENTS(X,SUBGRIDS,NDIMS,IERROR)
      IF (NDIMS.GT.3) WRITE(6,*)'dimension too high'
      DO IDIM=NDIMS+1,2*NDIMS
         SUBGRIDS(IDIM)=1
      END DO

C Get processor masks
      DO IDIM=1,NDIMS
         PMASKS(IDIM)=0
      END DO
      CALL CMF_GET_PROCESSOR_MASKS(X,PMASKS(NDIMS+1),NDIMS,
     .   IERROR)
```


Copyright © 1994 Thinking Machines Corporation

```
C Lay out the axes (first half :SERIAL, second half :NEWS)
      DO IDIM=1,NDIMS
         ORDERS(IDIM)=CMF_SERIAL_ORDER
         ORDERS(NDIMS+IDIM)=CMF_NEWS_ORDER
      END DO

C Determine axis extents
      DO IDIM=1,NDIMS
         EXTENTS(IDIM)=SUBGRIDS(IDIM)
         EXTENTS(IDIM+NDIMS)=2**POPCNT(PMASKS(IDIM+NDIMS))
      END DO

C Determine offsets
      DO IDIM=1,2*NDIMS
         OFFSETS(IDIM)=0
      END DO

C Form alias
      CALL CMF_ALIAS_DETAILED(ALIAS,2*NDIMS,EXTENTS,ORDERS,
     .   SUBGRIDS, PMASKS,OFFSETS,.FALSE.,X,IERROR)

      RETURN
      END


      SUBROUTINE STENCIL(NLOOP,X,COEF,M1,M2,N1,N2)

      DOUBLE PRECISION, DIMENSION(M1,M2,N1,N2) :: X, COEF
      DOUBLE PRECISION Y(0:M1+1,0:M2+1,N1,N2)
      DOUBLE PRECISION, DIMENSION(0:M2+1,N1,N2) :: GHOST1,GHOST2
      DOUBLE PRECISION, DIMENSION(0:M1+1,N1,N2) :: GHOST3,GHOST4
CMF$  LAYOUT
CMF   X(:SERIAL,:SERIAL,:BLOCK=1:PROCS=N1,:BLOCK=1:PROCS=N2)
CMF$  LAYOUT
CMF   COEF(:SERIAL,:SERIAL,:BLOCK=1:PROCS=N1,:BLOCK=1:PROCS=N2)
CMF$  LAYOUT
CMF   Y(:SERIAL,:SERIAL,:BLOCK=1:PROCS=N1,:BLOCK=1:PROCS=N2)
CMF$  LAYOUT GHOST1(:SERIAL,:BLOCK=1:PROCS=N1,:BLOCK=1:PROCS=N2)
CMF$  LAYOUT GHOST2(:SERIAL,:BLOCK=1:PROCS=N1,:BLOCK=1:PROCS=N2)
CMF$  LAYOUT GHOST3(:SERIAL,:BLOCK=1:PROCS=N1,:BLOCK=1:PROCS=N2)
CMF$  LAYOUT GHOST4(:SERIAL,:BLOCK=1:PROCS=N1,:BLOCK=1:PROCS=N2)

C Copy in bulk values
      Y=0.d0
```


Copyright © 1994 Thinking Machines Corporation

```
      Y(1:M1,1:M2,:,:)=X(1:M1,1:M2,:,:)
C Perform loop
      DO (NLOOP) TIMES

C        Load border values into dummy arrays

         GHOST1(0:M2+1,:,:)=Y(1,      0:M2+1,:,:)
         GHOST2(0:M2+1,:,:)=Y(M1,     0:M2+1,:,:)
         GHOST3(0:M1+1,:,:)=Y(0:M1+1,1,      :,:)
         GHOST4(0:M1+1,:,:)=Y(0:M1+1,M2,     :,:)

C        Move border values to nearby processors

         GHOST1=CSHIFT(GHOST1,DIM=2,SHIFT=+1)
         GHOST2=CSHIFT(GHOST2,DIM=2,SHIFT=-1)
         GHOST3=CSHIFT(GHOST3,DIM=3,SHIFT=+1)
         GHOST4=CSHIFT(GHOST4,DIM=3,SHIFT=-1)

C        Load border values from nearby procs into ghost cells

         Y(M1+1,   0:M2+1,:,:)=GHOST1(0:M2+1,:,:)
         Y(0,      0:M2+1,:,:)=GHOST2(0:M2+1,:,:)
         Y(0:M1+1,M2+1,   :,:)=GHOST3(0:M1+1,:,:)
         Y(0:M1+1,0,      :,:)=GHOST4(0:M1+1,:,:)

C        Perform stencil operation

         Y(1:M1,1:M2,:,:) = Y(1:M1,1:M2,:,:)
     .      +COEF(1:M1,1:M2,:,:)*(
     .      0.25d0*(
     .      Y(0:M1-1,1:M2,:,:)
     .      +Y(2:M1+1,1:M2,:,:)
     .      +Y(1:M1   ,0:M2-1,:,:)
     .      +Y(1:M1   ,2:M2+1,:,:)
     .      )
     .      -Y(1:M1,1:M2,:,:)
     .      )

      END DO

C Copy back values
      X(1:M1,1:M2,:,:)=Y(1:M1,1:M2,:,:)

      RETURN
      END
```


Copyright © 1994 Thinking Machines Corporation

Example 3

NOTE: This program is incompatible with "vector-length padding" of subgrids. Thus, it executes only on CM-5; if you use the vector units model, you must compile with the **cmf** switch **-nopadding**.

```
      PROGRAM DETAILED_ALIASES
      IMPLICIT NONE
      INTEGER N

      WRITE (6,*) 'enter N: '
      READ (5,*) N

      CALL BOTH(N)

      END


      SUBROUTINE BOTH(N)
      IMPLICIT NONE
      INCLUDE 'CMF_defs.h'

      INTEGER N
      REAL *8 A(N,N),B0(N,N),B1(N,N)
      INTEGER AALIAS(CMF_SIZEOF_DESCRIPTOR)
      INTEGER B1ALIAS(CMF_SIZEOF_DESCRIPTOR)
      INTEGER LEXTENTS(4),GEXTENTS(4),RANK,ASTAT
      INTEGER ORDERS(4),P1,P2,PMASKS(4),OFFSETS(4)
      INTEGER I,J
CMF$  LAYOUT A(:NEWS,:NEWS)
CMF$  LAYOUT B0(:NEWS,:NEWS)
CMF$  LAYOUT B1(:NEWS,:NEWS)
CMF$  LAYOUT AALIAS(:SERIAL)
CMF$  LAYOUT B1ALIAS(:SERIAL)
CMF$  LAYOUT LEXTENTS(:SERIAL)
CMF$  LAYOUT GEXTENTS(:SERIAL)
CMF$  LAYOUT ORDERS(:SERIAL)
CMF$  LAYOUT PMASKS(:SERIAL)
CMF$  LAYOUT OFFSETS(:SERIAL)
```


Copyright © 1994 Thinking Machines Corporation

```
      B0 = 0
      B1 = 0
      FORALL (I = 1:N, J = 1:N)
     $   A(I,J) = J + 1000*I

      CALL CMF_GET_LOCAL_EXTENTS(A,LEXTENTS,RANK,ASTAT)

      IF (RANK .NE. 2) THEN
         WRITE (6,*) 'Confusion: rank = ',RANK
         GOTO 999
      ENDIF

      GEXTENTS(1) = LEXTENTS(1)
      GEXTENTS(2) = LEXTENTS(2)

      ORDERS(1) = CMF_SERIAL_ORDER
      ORDERS(2) = CMF_SERIAL_ORDER
      ORDERS(3) = CMF_NEWS_ORDER
      ORDERS(4) = CMF_NEWS_ORDER

      CALL CMF_GET_PROCESSOR_MASKS(A,PMASKS,RANK,ASTAT)

      P1 = PMASKS(1)
      P2 = PMASKS(2)

      PMASKS(1) = 0
      PMASKS(2) = 0
      PMASKS(3) = P1
      PMASKS(4) = P2

      OFFSETS(1) = 0
      OFFSETS(2) = 0
      OFFSETS(3) = 0
      OFFSETS(4) = 0

      LEXTENTS(3) = 1
      LEXTENTS(4) = 1

      GEXTENTS(3) = 1
      GEXTENTS(4) = 1
      DO I=1,32
         PRINT *,P1,P2,GEXTENTS(1),GEXTENTS(2),GEXTENTS(3),
     $      GEXTENTS(4)
         IF (IAND(P1,1) .EQ. 1) THEN
            GEXTENTS(3) = GEXTENTS(3)*2
```


Copyright © 1994 Thinking Machines Corporation

```
         ENDIF
         IF (IAND(P2,1) .EQ. 1) THEN
            GEXTENTS(4) = GEXTENTS(4)*2
         ENDIF
         P1 = ISHFT(P1,-1)
         P2 = ISHFT(P2,-1)
         IF (P1+P2 .EQ. 0) EXIT
      END DO

      IF (N .NE. (GEXTENTS(1) * GEXTENTS(3))) THEN
         WRITE(6,*) 'length problem on axis 1'
         GOTO 999
      ENDIF
      IF (N .NE. (GEXTENTS(2) * GEXTENTS(4))) THEN
         WRITE(6,*) 'length problem on axis 2'
         GOTO 999
      ENDIF

      CALL CMF_ALIAS_DETAILED(AALIAS,4,GEXTENTS,ORDERS,LEXTENTS,
     $   PMASKS,OFFSETS,.TRUE.,A,ASTAT)

      CALL CMF_ALIAS_DETAILED(B1ALIAS,4,GEXTENTS,ORDERS,
     $   LEXTENTS,PMASKS,OFFSETS,.TRUE.,B1,ASTAT)

      B0 = A + CSHIFT(A,DIM=1,SHIFT=1)

      CALL FOO(B1ALIAS,AALIAS,LEXTENTS(1),LEXTENTS(2),
     $   GEXTENTS(3),GEXTENTS(4))

      IF (ANY(B0 .NE. B1)) THEN
         WRITE(6,*) 'FAIL'
      ELSE
         WRITE(6,*) 'PASS'
      ENDIF

999   CONTINUE
      RETURN
      END


      SUBROUTINE FOO(B,A,SG1,SG2,N1,N2)
      IMPLICIT NONE
      INTEGER SG1,SG2,N1,N2
      REAL *8 A(SG1,SG2,N1,N2),B(SG1,SG2,N1,N2)
      REAL *8 OLD_EDGE(1,SG2,N1,N2),NEW_EDGE(1,SG2,N1,N2)
```


Copyright © 1994 Thinking Machines Corporation

```
CMF$   LAYOUT A(:SERIAL,:SERIAL,:NEWS,:NEWS)
CMF$   LAYOUT B(:SERIAL,:SERIAL,:NEWS,:NEWS)
CMF$   LAYOUT OLD_EDGE(:SERIAL,:SERIAL,:NEWS,:NEWS)
CMF$   LAYOUT NEW_EDGE(:SERIAL,:SERIAL,:NEWS,:NEWS)

      OLD_EDGE(1,:,:,:) = A(1,:,:,:)
      NEW_EDGE = CSHIFT(OLD_EDGE,DIM=3,SHIFT=1)

      B(1:SG1-1,:,:,:) = A(1:SG1-1,:,:,:) + A(2:SG1,:,:,:)
      B(SG1,:,:,:) = A(SG1,:,:,:) + NEW_EDGE(1,:,:,:)

      RETURN
      END
```


Copyright © 1994 Thinking Machines Corporation