CM Fortran
2.2 Release Notes

Part I: Overview

1 Introduction

Version 2.2 offers new convenience features and greater robustness. This version also brings CM Fortran into conformance with Fortran 90 by updating some syntax and function interfaces that CM Fortran drew from an early draft of the standard. Some miscellaneous performance improvements are also provided.

The major features of Version 2.2 are:

- many new Fortran 90 intrinsic functions
- Fortran 90 compliance in previously non-compliant syntax
- bitwise logical operators (extensions beyond Fortran 90)
- improvements to global/local programming facilities
- ability to call CM Fortran subprograms from serial languages like Fortran 77 and C
- enhanced program checking and information from **`-safety`** and **`-list`** switches; enhanced **`INCLUDE`** statement and **`-I`** switch to permit directory searches for include files
- new symbols defined for the preprocessor **`cpp`**
- miscellaneous enhancements to the CM Fortran language and libraries

Copyright © 1994 Thinking Machines Corporation

2 Hardware and Software Requirements

2.1 Hardware Required

This release supports the Connection Machine models CM-5 and CM-5E and the following execution models:

- global vector units **`-vu`**
- global SPARC nodes **`-sparc`**
- nodal **`-vu -node`**
- global/local **`-vu ... -local`**
- CM Fortran simulator **`-cmsim`**

2.2 Software Required

This release requires the following or any later versions of CM-5 software:

- CMOST 7.3
- RTS 8.0.4
- CMMD 3.2 for nodal and global/local models

2.3 Software Supported

This release supports the following or later versions of CM layered software products:

- Prism 2.0
- CMSSL 3.2
- CM/AVS 1.1
- CMX11 1.6
- CMAX 1.0

Version 2.2, October 1994
Copyright © 1994 Thinking Machines Corporation

3 Compatibility Information

3.1 With Earlier Versions of CM Fortran

Compiling and linking:

- CM Fortran Version 2.2 does not require recompilation or relinking of programs linked under Version 2.1. However, to benefit from the most recent upgrades to the compiler and operating system, we recommend that you recompile and relink existing CM Fortran programs before executing them under 2.2.

- CM Fortran object modules (`.o` files) created under Version 2.1 can be linked together with object modules created under 2.2.

There are no incompatible changes between Versions 2.1 and 2.2.

Version 2.2 has made a number of syntax changes for conformance with Fortran 90 (see Section 4.1.3). However, the older syntax is also supported in this release, so existing programs do not require recoding. New programs you write should use the new syntax. It is not always possible to mix old and new syntax in a single program. See Section 4.1.3 for details.

3.2 With Other Products

- Fortran 77

  CM Fortran supports all of Fortran 77 (though not all the extensions added by other vendors). This serial code executes only on the partition manager, however. To take advantage of parallel processing, you need to translate Fortran 77 `DO` loops into CM Fortran (Fortran 90) array operations. The CMAX converter is useful for this purpose. See *Using the CMAX Converter*, Version 1.0 or 2.0.

- Fortran 90

  CM Fortran is a partial implementation of Fortran 90. The major Fortran 90 features *not* offered are:

  - derived types
  - modules

Copyright © 1994 Thinking Machines Corporation

- **TARGET** attribute and pointer assignment
- recursion
- array operations on **CHARACTER** arrays
- some string intrinsic functions
- storage and sequence association for arrays used in array operations ("CM arrays")
- implied **DO** loops in array constructors
- overloaded functions and operators
- function interface differences:
  CHAR function: no **KIND** argument
  ASSOCIATED function: no **TARGET** argument
  INDEX function: no **BACK** argument
- free source form
- user keyword arguments **PRESENT**, **OPTIONAL**
- non-advancing I/O

This release *does* supplement some non-compliant CM Fortran syntax with Fortran 90−compliant alternatives. See Section 4.1.3.

Copyright © 1994 Thinking Machines Corporation

4 New Features

This section summarizes the new and enhanced features in Version 2.2, as well as changes that improve performance.

4.1 New Functionality

4.1.1 New Fortran 90 Intrinsic Functions

Version 2.2 includes several new Fortran 90 intrinsic functions and subroutines. Optional arguments are enclosed in brackets. Full descriptions of these functions appear in Part II of these release notes.

- Numeric Functions

  These functions perform simple numeric operations:

```
CEILING( A )
FLOOR( A )
MODULO( A, P )
```

- Kind Functions

  These functions return the smallest kind needed to represent certain numbers:

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

- Numeric Inquiry Functions

  These functions return information about the system's numerical capabilities when arguments are represented as model numbers, as described in Section 4.1.2:

```
DIGITS( X )
EPSILON( X )
HUGE( X )
MAXEXPONENT( X )
MINEXPONENT( X )
PRECISION( X )
RADIX( X )
RANGE( X )
TINY( X )
```


Copyright © 1994 Thinking Machines Corporation

- Bit-Level Function

  This function returns information about the computer's representation of the argument's type:

```
BIT_SIZE( I )
```

- Floating-Point Manipulation Functions

  These functions perform various manipulations on floating-point numbers, including extraction and modification of specific fields:

```
EXPONENT( X )
FRACTION( X )
NEAREST( X, S )
RRSPACING( X )
SCALE( X, I )
SET_EXPONENT( X, I )
SPACING( X )
```

- Subroutines

  These new subroutines return system time information and handle random number generation:

```
DATE_AND_TIME([DATE] [,TIME] [,ZONE] [,VALUES])
RANDOM_NUMBER( HARVEST )
RANDOM_SEED( [ SIZE ] [, PUT ] [, GET ] )
SYSTEM_CLOCK( COUNT, COUNT_RATE, COUNT_MAX )
```

4.1.2 Models for Integer and Real Data

The representation of integer and real data can be described by two numerical models. These models are the basis for the information returned by the floating-point manipulation functions.

The model set for integer i is defined by

$$i = s \cdot \sum_{k=1}^{q} w_k \, 2^{k-1}$$

where q is a positive integer, each w_k is 0 or 1, and s is $+1$ or -1.

Copyright © 1994 Thinking Machines Corporation

The model set for real x is defined by:

$$x = \begin{cases} 0 \text{ or} \\ s \cdot 2^e \cdot \sum_{k=1}^{p} f_k \, 2^{-k} \end{cases}$$

where p is an integer exceeding one; each f_k is either zero or 1, except f_1 which is 1; s is $+1$ or -1; and e is an integer that lies in the range $e_{\max}$ to $e_{\min}$, inclusive. For $x = 0$, its exponent e and digits f_k are defined to be zero.

The integer parameter q determines the set of model integers, and the integer parameters p, $e_{\min}$, and $e_{\max}$ determine the set of model floating point numbers. The parameters of the integer and real models are available for each integer and real data type implemented by the processor. The parameters characterize the set of available numbers in the definition of the model. The numeric manipulation and inquiry functions provide values related to the parameters and other constants related to them.

4.1.3 Syntax Updates for Fortran 90 Compliance

CM Fortran previously implemented an early draft ("version 104") of Fortran 90. A few syntax changes occurred between that version and version 118, which became the ANSI and ISO standard. CM Fortran now provides standard Fortran 90 syntax as the recommended alternative to the older variants.

When possible, the CM Fortran compiler accepts either or both the new and old variants within a single program. When old and new variants are mutually exclusive, a new compiler switch, **-[no]f90syntax**, determines which set of variants is accepted.

-[no]f90syntax default: **-nof90syntax**

Controls the compiler's interpretation of arguments in certain intrinsic functions, including argument order when argument keywords are not used. The functions affected are **AINT**, **ANINT**, **REAL**, **CMPLX**, **RESHAPE**, **CSHIFT**, and **EOSHIFT**. With the negative form, the default, the compiler interprets function references as in previous releases of CM Fortran (based on an early draft of Fortran 90); with the positive form, the compiler

Copyright © 1994 Thinking Machines Corporation

interprets function references as defined in the Fortran 90 standard. The default will change in a future version.

For the following features, the 2.2 CM Fortran compiler accepts both variants. We recommend the Fortran 90 variant, but will continue to support the other for back-compatibility.

Fortran 90 (V118)	Older Variant (V104)
/= operator	**<>**
DIMENSION attribute	**ARRAY**
SIZE intrinsic	**DSIZE**
SHAPE intrinsic	**DSHAPE**
UBOUND intrinsic	**DUBOUND**
LBOUND intrinsic	**DLBOUND**
DOT_PRODUCT intrinsic	**DOTPRODUCT**

For the following features, the CM Fortran compiler accepts one set of variants *or* the other, under control of the switch **-f90syntax**. The default is **-nof90syntax** to guarantee that existing programs will compile. However, we recommend that you write new code using only the Fortran 90 variants and compile with **-f90syntax**. The Fortran 90 variants are more portable to other systems and will become the default in a future version of CM Fortran.

Version 2.2, October 1994
Copyright © 1994 Thinking Machines Corporation

Fortran 90 `-f90syntax`	Older CMF `-nof90syntax`	Remarks
`AINT(A [,KIND])`	`AINT(A)`	optional arguments not accepted under `-nof90syntax`
`ANINT(A [,KIND])`	`ANINT(A)`	optional arguments not accepted under `-nof90syntax`
`REAL(A [,KIND])`	`REAL(A [,MOLD])`	second argument keyword changed
`CMPLX(X [,Y] [,KIND])`	`CMPLX(X [,Y] [,MOLD])`	third argument keyword changed
`RESHAPE(SOURCE,SHAPE[,PAD][,ORDER])`	`RESHAPE(MOLD,SOURCE[,PAD][,ORDER])`	order of first two arguments and one keyword changed
`CSHIFT(ARRAY,SHIFT[,DIM])`	`CSHIFT(ARRAY,DIM,SHIFT)`	argument order when invoked without keywords; `DIM` optional
`EOSHIFT(ARRAY,SHIFT[,BOUNDARY][,DIM])`	`EOSHIFT(ARRAY,DIM,SHIFT[,BOUNDARY])`	argument order when invoked without keywords; `DIM` optional

Copyright © 1994 Thinking Machines Corporation

4.1.4 Bitwise Logical Operators

Version 2.2 introduces six logical operators that operate bitwise on integers. Note that these operators are not part of Fortran 90.

.AND.	Performs the bitwise logical conjunction of two integer values.
.XOR.	Performs the bitwise logical exclusive disjunction of two integer values.
.OR.	Performs the bitwise logical inclusive disjunction of two integer values.
.NOT.	Performs the bitwise logical complement of an integer value.
.NEQV.	Performs the bitwise logical exclusive disjunction of two integer values.
.EQV.	Performs **NOT**(the bitwise logical exclusive disjunction of two integer values).

These operators correspond to bit-level intrinsic functions as follows:

Operator	Function
.AND.	**IAND**
.OR.	**IOR**
.XOR.	**IEOR**
.NOT.	**NOT**
.NEQV.	**IEOR**
.EQV.	**NOT(IEOR)**

These operators have the same precedence as the logical operator. That is, **I+2 .AND. J** is equivalent to **IAND(I+2,J)**.

Copyright © 1994 Thinking Machines Corporation

4.1.5 Improvements to Global/Local Programming Facilities

The appendix of these release notes contains an update of the chapter on global/local programming since the Version 2.1 *Programming Guide* and *Libraries Reference Manual.* This section summarizes the changes in Version 2.2.

The following improvements are now implemented:

- The prototypes in **`.proto`** files may contain newlines.
- Prototypes should end in a semi-colon, to permit the full generality of the newline feature. This version warns when encountering a prototype that does not end in `;`. Future versions will require it.
- It is now possible to compile a local CM Fortran routine with **`-nopadding`**.
- The global arrays passed to a C local routine are now treated as 0-based, and local index arguments to the C inquiry functions assume a 0-based array.
- A new inquiry procedure, **`CMGL_GLOBAL_TO_PHYSICAL`** identifies the processing node and location where a given element of a global array is allocated. See Part II of these release notes for a full description.

  CM Fortran Syntax:

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

  C Syntax:

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

  The C function returns the node number; notice the different calling sequence in the CM Fortran soubroutine, which returns values in the **`PN`** and **`L_INDEX`** arguments.

NOTE: The 2.1 documentation points out that you cannot cmprofile the local part of a global/local program. You can, however, profile the global part. Prism then shows how much total time is spent in a local routine, but you cannot determine how that time is being spent. For this purpose, you compile the global part of the code with **`-cmprofile`**, compile the local part without **`-cmprofile`**, and then link it all together.

Copyright © 1994 Thinking Machines Corporation

4.1.6 Calling CM Fortran Subprograms from Serial Languages

CM Fortran now provides the utility library procedure **CMF_INIT**, which initializes the parallel processing unit. You must call this procedure before calling any CM Fortran subprogram from a program compiled with a compiler other than **cmf**. This requirement applies to programs that run globally (on the CM-5 partition manager) or locally (on the individual nodes).

The **CMF_INIT** procedure is useful for calling CM Fortran kernels from serial Fortran, C, or other languages with C-compatible calling conventions. Both Fortran and C versions of the initialization procedure are provided; see Part II of these release notes for full information. To call CM Fortran from C*, please see the instructions given in the *C* User's Guide*.

Fortran Syntax:

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

C Syntax:

```
void CMF_init (int padding, int axisreorder)
```

4.1.7 Improvements to Compiler Switches

Enhanced Safety Checking

The run-time safety checks performed with **-safety=1** or any higher value now include conformance checking (shape and layout) between actual and dummy arguments and checking of array-valued functions against the interface block in the calling program.

-safety=*number* default: **-safety=0**

Causes code to be generated that performs some run-time checking. At level **safety=0** (zero), no run-time checking is performed. At level **safety=1**, code is generated that detects some mismatches between actual and dummy arguments and also compares array-valued functions with the results described in the interface block in the calling program. In particular, the run-time code checks that the number of actual arguments equals the number of dummy arguments, the home of each actual argument is the same as the home of the dummy argument, the rank of each CM actual array argument is the

Version 2.2, October 1994
Copyright © 1994 Thinking Machines Corporation

same as the dummy argument, the shape (the extents in rank order) of each CM actual array argument is the same as the dummy argument, and the ordering (**:NEWS** or **:SERIAL**) of each axis of each CM actual array argument is the same as the dummy argument. Comparable checks are made of array-valued functions. At level **safety=10** all run-time safety checks are enabled. Currently these are: argument and function checking (as listed above), use of uninitialized CM arrays, and out-of-bounds vector-valued subscripts. Note that the generated code does not check the types of arguments or of array-valued functions.

Expanded Searching for INCLUDE Files

The CM Fortran **INCLUDE** line has been enhanced to accept relative pathnames, as well as absolute pathnames, for files to be included. If the locations of standard CM system include files like **CMF_defs.h** are customized on your site, you may want to change to a relative pathname. For example, change an **INCLUDE** line from

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

to

```
INCLUDE 'CMF_defs.h'
```

In addition, the compiler switch **-I** enables the user to specify paths.

-I*dir*

Adds the directory *dir* to the list of directories to be searched for include files specified with relative pathnames (pathnames not beginning with "/"). Include files are sought first in the current directory, then in **-I** paths in the order specified by the user on the **cmf** command line, then in **-I** paths specified at installation time, then in the directory containing the source file (if different from current directory), and finally in the default directories under **/usr/include**.

The user-specified **-I** switches are passed through to the C compiler **cc** and preprocessor **cpp**, which treat them in their own way. The **cmf** compiler does not pass **-I** paths specified at CM Fortran installation time through to **cc** or **cpp**.

Copyright © 1994 Thinking Machines Corporation

Enhanced Line Numbering with List

Previous versions of the CM Fortran **-list** switch could not supply source line numbers for code compiled with **-O**. The generated **.lis** file showed **"null stream"** instead of communication routines because it could not determine where in the program the optimized code originated.

```
COMMUNICATION ROUTINES
  Name                  Line Number (number of times)
  SUM (into scalar)     [<<null stream>>: 0(2)]
  SUM (into vector)     53, 73
```

Version 2.2 does supply line numbers in listings of optimized code. If the line number is approximate, it is marked with an asterisk. The output now looks like:

```
COMMUNICATION ROUTINES
  Name                  Line Number (number of times)
  SUM (into scalar)     73*(2)
  SUM (into vector)     53, 73
```

Compiler now passes −D switch to f77 compiler

This release of the compiler passes switches **-D***name* or **-D***name***=***def* to the **f77** compiler when processing a **.F** file.

In previous releases, the compiler passed **-D** switches to **cpp** but not to **f77**. It was necessary to use **-Z** to accomplish this:

```
% cmf -Zf77 "-DHELLO" myfile.F
```

The use of **-Z** for this purpose is no longer necessary.

Copyright © 1994 Thinking Machines Corporation

4.1.8 New Symbols Defined for cpp

When processing **.FCM** files, this version of the compiler passes to **cpp** certain predefined symbols that identify the target machine, execution model, and several switches. This feature allows use of conditional **#ifdef** code that depends on compiler switches.

Symbols are now defined that

- identify the target machine:

  -cm5 -DCM5
  -cmsim -DCMSIM [Note: **-cmsim** not available in 2.2 Beta]

- identify the target execution model:

  -sparc -DCMF_SPARC
  -vu -DCMF_VU
  -node -DCMF_NODE
  -local -DCMF_LOCAL (for **-local** code only)

- identify use of certain switches that may affect performance or introduce incompatibilities between program units if not uniform:

  -O -DCMF_OPTIMIZE
  -g -DCMF_DEBUG
  -cmdebug -DCMF_CMDEBUG
  -cmprofile -DCMF_CMPROFILE
  -padding -DCMF_PADDING
  -axisreorder -DCMF_AXISREORDER

4.1.9 Miscellaneous Enhancements

- CM Fortran arrays may be any rank up through 31.

- The utility library procedure **CMF_ALIAS_TYPE** now accepts the **TYPE** argument **CMF_LONG_S_INTEGER** (**-vu** model only).

- The **ALLOCATE** and **DEALLOCATE** statements now conform to Fortran 90 in accepting the **STAT=** specifier.

  The specifier is a scalar integer variable. If present, this variable is defined with the value of zero upon successful execution of the **ALLOCATE** or **DEALLOCATE** statement; any positive value indicates

Copyright © 1994 Thinking Machines Corporation

failure. (If the allocation or deallocation operation fails in the absence of a **STAT=** specifier, program execution terminates.)

```
INTEGER N,M,STAT_VAR
REAL, POINTER, DIMENSION(:,:) :: A
...
ALLOCATE (A(N,M), STAT=STAT_VAR)
...
DEALLOCATE (A, STAT=STAT_VAR)
```


Version 2.2, October 1994
Copyright © 1994 Thinking Machines Corporation

4.2 Performance Enhancements

4.2.1 Linking with CMSSL

Certain CM Fortran procedures give significantly better performance if the program is linked with the CM Scientific Software Library, CMSSL:

```
% cmf myprogram.fcm -lcmssl
```

Beginning with CMSSL 3.2 and CM Fortran 2.2 Beta, performance is improved for:

RANDOM_NUMBER intrinsic
CMF_RANDOM_LONG_S_INTEGER utility procedure

As in previous releases, performance is improved for the following when CMSSL is linked in:

MATMUL
CMF_RANDOM
CMF_RANK
CMF_SORT
CMF_ORDER

This information has been added to the individual man pages for these procedures.

4.2.2 Using FORALL

Several restrictions on the performance of **FORALL** have been removed in Version 2.2.

FORALL no longer executes serially if an index variable is used in the following expressions:

- multiple-term array constructor, such as `[B(I),C(I),D(I)]`
- multiple-dimension expression to **MAXLOC** or **MINLOC**, such as `MAXLOC(B(I,:,:))`

The compiler now recognizes when a **FORALL** upper bound is the same as the declared upper bound for an axis, thereby improving performance.

See Section 7.1 for the remaining restrictions on **FORALL**.

Copyright © 1994 Thinking Machines Corporation

4.2.3 Transparent Performance Enhancements

- The compiler now makes greater use of stack space for compiler-generated temporaries, instead of using heap space exclusively.
- Gather/scatter operations by assignment are generally faster than before.
- Copy propagation for local arrays has been improved.
- The compiler performs more code motion and code block mergers to increase the size of the PN code blocks generated.
- Reductions along serial axes are now faster.

Copyright © 1994 Thinking Machines Corporation

5 Changes from Previous Versions

5.1 Changes to Interface

No incompatible changes.

5.2 Deprecated or Obsolete Features

None.

5.3 Obsolete Features Removed

None.

Copyright © 1994 Thinking Machines Corporation

6 Notice of Possible Future Changes

6.1 Phasing Out Linking with Sun f77 Libraries

CM Fortran automatically links with Sun libraries if they are installed on the system. This behavior will be discontinued in later versions of CM Fortran. At that point, it will be necessary to specify Sun libraries on the **cmf** command line if you wish to link with them.

Even in the current version, you can get a run-time error for linking with Sun's dynamically bound libraries. When a program that was linked with these libraries does not find them at run time, the following error is signaled:

```
ld.so: libF77.so.1: not found
```

You can prevent this problem by linking statically by means of the **-Bstatic** switch, which **cmf** passes on to the linker:

```
% cmf myfile.fcm -Bstatic
```

6.2 cmf77 Procedure FLUSH Deprecated

Future releases of CM Fortran will not support the **cmf77** library procedure **FLUSH**. We recommend that you remove calls to **FLUSH** from CM Fortran programs. Removal has no effect, since CM Fortran now flushes I/O buffers automatically after every write operation (not just at program exit).

Nodal programs that call **FLUSH** and also include Fortran 77 modules may encounter a linking error, since the procedure is also defined in Sun libraries. Removing **FLUSH** from the CM Fortran code avoids the error.

6.3 Axis Ordering in Memory

The current default axis ordering, **-axisreorder**, exists in the compiler for historical reasons. The **-noaxisreorder** switch produces the standard Fortran left-to-right ordering. In the future, the left-to-right ordering may become the default. We encourage its use.

Copyright © 1994 Thinking Machines Corporation

NOTE: **`-axisreorder`** affects only CM arrays. Front-end array axes are always stored in left-to-right order, regardless of the setting of this switch.

6.4 Vector-Length Padding

Similarly, the current default padding of the (non-serial) subgrid to a multiple of vector length also exists for historical reasons. (The compiler did not originally generate clean-up code for loops in which the number of times the loop is executed is not a multiple of the vector length.) In the future, **`-nopadding`** may replace **`-padding`** as the default. We encourage its use.

6.5 Compatibility

The previous defaults for **`-[no]axisreorder`** and **`-[no]padding`** have been retained in this release for back-compatibility. If the defaults change, or if you begin to switch over to the nondefault options, consider compatibility issues like the following:

- Suppose you have a library compiled under the current default **`-axisreorder`**. In the future, if the default changes, you may try to call this library from a newly compiled procedure (with **`-noaxisreorder`**). The program will fail because all program units in a program must be compiled with the same setting of this switch.

- Suppose you have an array **`A`** laid out **`(:SERIAL,:NEWS)`** and you pass **`A(1,:)`** to a subroutine. With the current default **`-axisreorder`**, this section is passed in place. In the future, with the **`-noaxisreorder`** default, the section would first be sent to a temporary location, thus slowing down the program.

Copyright © 1994 Thinking Machines Corporation

7 Limitations and Restrictions

7.1 Temporary Restrictions

FORALL has several temporary inefficiencies, where it either executes serially or fails to generate the most efficient parallel instructions.

- Temporarily, **FORALL** executes serially if it references:
  - the intrinsic function **RESHAPE** with a **PAD** argument, with triplet **SOURCE**, or with **MOLD** or **ORDER** arguments not specified with literal constants
- Temporarily, **FORALL** executes serially if an index variable is used in any of the following expressions:
  - front-end array expression
  - unaligned binary triplet expression, such as `B(0:I) + B(I:2*I)`
- Temporarily, **FORALL** does not generate the most efficient parallel instructions for array transfers between the parallel processors and the control processor. A simple CM-to-FE transfer operation is expressed as:

  ```
  FORALL (I=1:N) FE(I) = CM(I)
  ```

  In the current release, this statement generates a serial **DO** loop with a **READ_TO_PROCESSOR** or **WRITE_FROM_PROCESSOR**. It is better to use the utility procedures **CMF_FE_ARRAY_TO**/**FROM_CM**.

7.2 Permanent Restrictions

The following cause **FORALL** to execute serially:

- Reference to a character variable, such as `STRING(I)`.
- Reference to an external function.

Version 2.2, October 1994
Copyright © 1994 Thinking Machines Corporation

- "Too many" triplet axes, such as `A(1:I,...,1:L)`. The operation executes serially if the number of vector axes (those with colons) plus the number of distinct **FORALL** indices referenced is above a certain threshold. The threshold in the current release is 31.

- **CSHIFT** or **EOSHIFT** of a triplet axis using a **FORALL** index, such as
  `CSHIFT( B(1:I),1,1 )`.

- A nonconstant **DIM** argument, that is, a dimension not known at compile time, to any reduction function, or to the transformational functions **CSHIFT** or **EOSHIFT**, **FIRSTLOC** or **LASTLOC**, **PROJECT**, **REPLICATE**, **SPREAD**, or to the inquiry functions **UBOUND**, **LBOUND**, or **SIZE**.

Another restriction is that **FORALL** does not support assumed-size arrays (those declared with * as the last axis extent) or the type **CHARACTER**.

7.3 Restrictions Removed

With CM Fortran 2.2 and CMSSL 3.2, it is no longer necessary to compile CM Fortran programs with **-axisreorder** and/or **-padding**. CMSSL now supports the negative forms as well.

Copyright © 1994 Thinking Machines Corporation

8 Documentation for This Release

- User manuals

  This release consists of these release notes, along with the CM Fortran Version 2.2 documentation set:

  - *CM Fortran Language Reference Manual*
  - *CM Fortran Libraries Reference Manual*
  - *CM Fortran User's Guide*
  - *CM Fortran Programming Guide*
  - *CM-5 CM Fortran Performance Guide*
  - *CM Fortran Array Operations Quick Reference Guide*
  - *Getting Started in CM Fortran*, October 1994

  All the Version 2.2 manuals are available in hardcopy.

- The following are provided in CMview document collections CM5-2 and later:

  - On-line viewing and search of all the documents except release notes and *Quick Reference Guide*.
  - PostScript versions of all the documents (except *Quick Reference Guide*), which are intended for printing. These PostScript documents are typically installed on CM-5 systems under **`/usr/doc/cmfortran`**.

- ASCII versions of the release notes and the manuals can be accessed through Prism.

- Man pages

  This beta release provides up-to-date on-line man pages for:

  - **`cmf`** command
  - all intrinsic functions
  - all procedures in the CM Fortran libraries:
    - utility library
    - global/local library
    - **`cmf77`** library

Copyright © 1994 Thinking Machines Corporation

View the procedure man pages with the command **man**, followed by the procedure name in uppercase. (The utility and global/local procedures can be specified in lower- or mixed case, but the intrinsics and **cmf77** routines must be specified in uppercase to avoid name conflicts with Sun man pages.)

- Directories of sample programs

  On-line examples appear under **/usr/examples/cmfortran**. If you do not find them there, ask your site system adminstrator where they have been installed at your site.

Copyright © 1994 Thinking Machines Corporation

9 Errors, Feedback, and Assistance

9.1 Bugupdate Files

The status of known bugs can be found in **/usr/doc/cmfortran-2.2.bugupdate**, which is accessible through Prism. Check with your site system adminstrator if this file has been moved.

9.2 Documentation Corrections

9.2.1 Global/Local Library Function CMGL_global_to_local

CM Fortran documentation for Version 2.1 erroneously described the global/local library function **CMGL_global_to_local**. The description should read, "Given global array indices, provides local array indices." In addition, both the **L_INDEX** and **G_INDEX** arguments must be serial arrays.

The correct syntax is:

CM Fortran Syntax:

```
 CALL CMGL_GLOBAL_TO_LOCAL( ARRAY, G_INDEX, L_INDEX,
&                           LOCAL )
```

C Syntax (declaration):

```
void CMGL_global_to_local( CMRT_desc_t ARRAY,
            int *G_INDEX, int *L_INDEX, int *LOCAL )
```

The line at the top of the man page should say:

Given global array indices, provides local array indices.

The changes from the previously published documentation are:

- one-line description: reversed the words "local" and "global"
- added **LOCAL** argument to C syntax
- fixed function name in C syntax line

Version 2.2, October 1994
Copyright © 1994 Thinking Machines Corporation

9.2.2 Setting of −[no]padding

CM Fortran documentation for Version 2.1 stated that all program units must be compiled with the same setting of **-[no]padding**. This is not correct. Only the setting used in compiling the main program unit is used, that is, passed to the run-time system to govern array layout decisions. Contrary settings in other program units are ignored.

For programs compiled without a CM Fortran main program, the operative setting is governed by an argument to the procedure **CMF_INIT**.

9.2.3 Code block size may differ with −O

The performance guide states (in Section 7.2.2) that consecutive array expressions that reference arrays of different shape are put in separate code blocks. Beginning with Version 2.1, the optimizer (**-O**) may reorder computation to increase code block size. As a result, it is often difficult to tell what operations will and will not break a code block without a very detailed understanding of the compiler's internals.

9.2.4 CMF_RANK, CMF_SORT, and CMF_ORDER Do Not Take COMPLEX arguments

The Version 2.1 documentation for **CMF_RANK**, **CMF_SORT**, and **CMF_ORDER** erroneously states that these procedures work for arrays of any type. In fact, they do not support types **COMPLEX** or **DOUBLE COMPLEX**.

9.3 Request for Feedback

Please contact your Thinking Machines on-site applications engineer to report bugs or provide feedback about CM Fortran 2.2 software or documentation.

Copyright © 1994 Thinking Machines Corporation