CM Fortran
2.3 Release Notes

1 Introduction

This release of CM Fortran resolves a number of limitations reported in prior releases of CM Fortran. It also introduces one experimental facility, which is explained in Section 6 below.

2 Hardware and Software Requirements

2.1 Hardware Required

The CM Fortran 2.3 release runs on all models of the CM-5 system and supports the following execution models:

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

Copyright © 1996 Thinking Machines Corporation

2.2 Software Required

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

- CMOST 7.4 Final
- CMRTS 8.1
- 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

3 Compatibility Information

3.1 With Earlier Versions of CM Fortran

Object files produced by earlier versions of CM Fortran are not compatible and must be recompiled and linked using the CM Fortran 2.3 driver and libraries.

3.2 With Other Products

3.2.1 Fortran 77

CM Fortran supports all of Fortran 77 (though not all extensions added by other vendors). This serial code executes only on the partition manager,

Copyright © 1996 Thinking Machines Corporation

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.

3.2.2 Fortran 90

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

- derived types
- modules
- **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

Copyright © 1996 Thinking Machines Corporation

4 Restrictions

FORALL behaves inefficiently in some circumstances — i.e., it either executes serially or simply generates inefficient parallel instructions. These circustances are described below.

- **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
  - a character variable, such as `STRING(I)`
  - an external function
- **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)`
- The following cause **FORALL** to execute serially:
  - The number of vector axes (those with colons) plus the number of distinct **FORALL** indices referenced exceeds 31
  - **CSHIFT** or **EOSHIFT** of a triplet axis using a **FORALL** index, such as `CSHIFT( B(1:I),1,1 ).`
  - A **DIM** argument that is not a constant — 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**
- **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)
```


Copyright © 1996 Thinking Machines Corporation

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**.

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

5 Documentation for this Release

All CM Fortran 2.2 documentation continues to apply to this release.

6 Experimental Facilities

This release introduces an experimental facility for garbage−collecting array geometries used by the run−time system. Application codes that create a very large number of distinct array geometries (e.g., that create arrays in a very large number of distinct sizes) suffer memory consumption as geometries are interned. If geometries are ephemeral, users may wish to garbage−collect geometries that are no longer associated with allocated array objects. Four utility library routines are introduced with release 2.3 for this purpose:

```
CMF_GEOMETRY_GC()
```

If geometry collection is enabled, this routine causes unreferenced geometries created while collection was enabled to be deallocated on both the control processor and the processing nodes. Otherwise, it is a null operation.

```
CMF_GEOMETRY_GC_ENABLE()
```

Calling this routine causes all geometries created thereafter to be enrolled for possible future garbage collection.

```
CMF_GEOMETRY_GC_DISABLE()
```


Copyright © 1996 Thinking Machines Corporation

Calling this routine causes the run−time system to cease enrolling geometries as collection candidates and disables the collective function of the **CMF_GEOMETRY_GC** routine.

```
CMF_GEOMETRY_GC_STATS(removed,notremoved,noncandidates)
```

Calling this routine after a call to **CMF_GEOMETRY_GC** reports statistics on the results of a collection operation. The integer output arguments *removed*, *notremoved* and *noncandidates* report, respectively, the number of geometries that were deallocated, the number of candidate geometries not deallocated due to an extant reference, and the number of interned geometries that were not candidates for collection.

If any problems are encountered in the use of this experimental facility, please see section 7.2 below.

7 Errors, Feedback, and Assistance

7.1 Bugupdate Files

The status of known bugs can be found in **/usr/doc/cmfortran-2.3.bugupdate**, which is accessible through Prism. Check with your site system adminstrator if this file has been moved.

7.2 Request for Feedback

Please contact your Thinking Machines on-site applications engineer to report bugs or provide feedback about CM Fortran 2.3 software or documentation.

Copyright © 1996 Thinking Machines Corporation

7.3 Documentation Corrections

7.3.1 Getting Started in CM Fortran

On page 28 of the *Getting Started in CM Fortran*, October 1994 manual, the FORALL statement in the middle of the page (between the two matrix diagrams) contains two errors.

- The *index−variable−name* "I" for the first *forall−triplet−spec* is also referenced as the subscript of the second *forall−triplet−spec*.
- The mask component of the *forall−triplet−spec* is missing.

To correct these errors, change

```
FORALL (I=1:N, J=1:I-1) H(I,J) = H(I,J+1)
```

to

```
FORALL (I=1:N-1, J=1:N-2, I>J) H(I,J) = H(I,J+1)
```

7.3.2 CM Fortran User's Guide

On page 21 of the *CM Fortran User's Guide*, Version 2.2, the description of **-safety=10** safety level is incorrect. Uninitialized array checking is *not* done; program memory is *not* initialized to all ones. Instead, the description on page 21 should read

10 Provides all of the functionality of levels 1−9, plus run-time checking for invalid addresses in communications operations.

This error is repeated on page 13 of the CM Fortran, Version 2.2, Release Notes.

Copyright © 1996 Thinking Machines Corporation

7.3.3 CM Fortran Programming Guide

On page 165 of the *CM Fortran Programming Guide* Version 2.2, the example of a segmented scan using FORALL contains the incorrectly formed expression **`MAXVAL([1:I],SEGMENT[1:I])`**. To correct this error, change

```
MAXVAL([1:I],SEGMENT[1:I])
```

to

```
MAXVAL([1:I],MASK=SEGMENT[1:I])
```

There is also the need for a cautionary note regarding use of assumed-shape dummy arrays (discussed in Section 6.2.2 of the programming guide).

NOTE

For multiple arrays with conformable shapes, use explicit dimensions rather than assumed shapes wherever possible as this will ensure more efficient compiler output. In the following example, subroutine **`SUB2`** will provide approximately twice the performance of **`SUB1`**; the additional information it contains about the shapes of **`X`** and **`Y`** informs the compiler explicitly that they are conformable. The compiler has insufficient information to make that determination about **`SUB1`**.

```
SUBROUTINE SUB1(X,Y)
REAL X(:), Y(:)
X+Y
END
```

```
SUBROUTINE SUB2(X,Y,N)
INTEGER N
REAL X(N), Y(N)
CMFS LAYOUT X(:NEWS), Y(:NEWS)
X+Y
END
```

7.3.4 CM Fortran Language Reference Manual

The following errors exist in the *CM Fortran Language Reference Manual* Version 2.2.

- Page 27, near bottom — In the example of double-precision real literal constants, one zero is missing from the decimal representation of the first value. To correct this error, change

  `0.00045` to `0.000045`

- Page 80, line 8 in NOTE — **`V`** is defined as having a shape of **`[4]`**, but is assigned only two values. To correct this error, change

Copyright © 1996 Thinking Machines Corporation

```
V = [3,5]
```
to
```
V = [2,3,4,5]
```

- Page 80, line 11 in NOTE — The **J** shown in the first bullet is wrong. To correct this error, change

```
A(V,J)
```
to
```
A(V,1)
```

- Page 81, *array−constructor−values* examples — The first example is incorrectly identified as an integer vector. To correct this, change

```
[1,(2,0),(0.,1.)]   ! an integer vector of length 3
```
to
```
[1,(2,0),(0.,1.)]   ! a complex vector of length 3
```

- Page 362 — The result value of **DPROD(X,Y)** is incorrectly described as **DBLE(X*Y)**. The actual result is **DBLE(X)*DBLE(Y)**. To correct this error, change

```
DBLE(*Y)
```
to
```
DBLE(X)*DBLE(Y)
```

7.3.5 CM Fortran Libraries Reference Manual

The **IOSTAT** argument description on the **CMF_CM_ARRAY_FROM_FILE_SO** manual page is incorrect. It currently states that a positive return value indicates success and specifies the number of bytes read, a zero indicates an end-of-file condition, and a negative value indicates failure. The correct definition of the **IOSTAT** argument is as follows:

IOSTAT — An INTEGER*4 variable whose value indicates the number of bytes that were read into the array specified by the **DEST** variable. Ordinarily, this will be all of the bytes in the source file. In some cases, however, the dimensions of the destination array are such that the source file cannot be equally distributed among its elements; i.e., the source file byte count is not an integer multiple of the **DEST** array dimensions. In such cases, the value returned by **IOSTAT** will be smaller than the source file byte count, but will still represent the number of bytes actually read into the destination array. Because distribution of those bytes is not generally predictable in a useful way, users are advised to treat

Copyright © 1996 Thinking Machines Corporation

any **IOSTAT** value that does not match the size of the entire array as an error condition.

Copyright © 1996 Thinking Machines Corporation