The
Connection Machine
System

CM Fortran Release Notes

Version 2.1 Beta 0
December 1992

Thinking Machines Corporation
Cambridge, Massachusetts

First printing, December 1992

****

PRELIMINARY DOCUMENTATION

The information in this document is subject to change without notice and should not be construed as a commitment by Thinking Machines Corporation. Thinking Machines assumes no liability for errors in this document.

This document does not describe any product that is currently available from Thinking Machines Corporation, and Thinking Machines does not commit to implement the contents of this document in any product.

****

Thinking Machines Corporation
245 First Street
Cambridge, Massachusetts 02142-1264
(617) 234-1000

Contents

Field Test Support

Field test software users are encouraged to communicate with Thinking Machines Corporation as fully as possible throughout the test period. Please report any errors you may find in this software and suggest ways to improve it.

When reporting an error, please provide as much information as possible to help us identify the problem. A code example that failed to execute, a session transcript, the record of a backtrace, or other such information is extremely helpful in this regard.

If your site has an applications engineer or a local site coordinator, please contact that person directly for field test support. Otherwise, please contact Thinking Machines' home office customer support staff:

Internet Electronic Mail:	`customer-support@think.com`
uucp Electronic Mail:	`ames!think!customer-support`
U.S. Mail:	Thinking Machines Corporation Customer Support 245 First Street Cambridge, Massachusetts 02142-1264
Telephone:	(617) 234-4000

CM Fortran Release Notes

1 About Version 2.1 Beta 0

This initial beta release of CM Fortran Version 2.1 provides the following new functionality for CM-5 systems with vector units:

- Support for the CM file system on the Scalable Disk Array.
- Support for the 64-bit integer data type in the CM Fortran language and Utility Library

Porting Code from V2.0

Version 2.0 programs should be recompiled and relinked to execute under Version 2.1 Beta 0.

Notes on Beta 0

- This beta release does not support UNIX profiling. Use of the `cmf` compiler option `-pg` causes a link-time error.
- The compiler issues a warning when asked to invoke a foreign compiler or to link with Sun libraries, since this behavior will be discontinued in a later beta release of V2.1. In the current release, however, it does still perform the requested operation.

2 Using the Scalable Disk Array

The CM-5 system now supports two CM file systems, in addition to the UNIX file system:

- The new SFS, or Scalable File System, which resides on the Scalable Disk Array.
- The CMFS, or CM (DataVault) File System, which resides on storage devices on the CMIO bus, including the DataVault, VMEIO host computer, and CM-HIPPI.

See the CM-5 I/O documentation for more information on the CM file systems.

The I/O procedures in the CM Fortran Utility Library support both CM file systems, transparently to the user.

For systems with more than one kind of storage device installed, the device affected by a call to a CM Fortran I/O procedure is governed by a site default or by the user environment variable **`CMFS_PATHTYPE`**. For information on the defaulting behavior, see the CMOST man page for **`CMFS_PATHTYPE`** or the CM I/O documentation.

3 Using 64-Bit Integers

This version provides a new data type, the 64-bit integer, both for scalar data on the partition manager and for parallel data on the vector units. This type will not be supported on CM systems without vector units.

3.1 Type Specifications

CM Fortran now supports seven types for scalar and parallel data on CM-5 systems with vector units (along with character type for scalars only). The seven types, along with their lengths in bits and their predefined symbolic names are shown below.

Data Type	Length	Symbolic Name
LOGICAL	32	(none)
INTEGER*4	32	_SINGLE_INT_
INTEGER*8	64	_DOUBLE_INT_
REAL*4	32	_SINGLE_
REAL*8	64	_DOUBLE_
COMPLEX*8	32	_SINGLE_COMPLEX_
COMPLEX*16	64	_DOUBLE_COMPLEX_

DOUBLE PRECISION (see REAL*8)
DOUBLE COMPLEX (see COMPLEX*16)

Integer and floating-point values default to the 32-bit length. The `cmf` compiler provides an option, `-double_precision`, that causes real values only to default to the 64-bit length.

You can declare the desired length of a variable by using the CM Fortran language extension *type*length*. For example:

```
INTEGER*8 A
COMPLEX*16 Z
```

Alternatively, you can use the Fortran 90 *kind* mechanism, described below.

KIND Keyword

Fortran 90 describes the keyword **KIND**, which associates a length with the type in a specification statement. The value of **KIND**, called the *kind type parameter*, must be a scalar integer *initialization expression*, meaning a compile-time constant. In CM Fortran, it can be any constant expression that evaluates to 32 or 64. For example:

```
INTEGER(KIND=64) A
INTEGER(KIND=_DOUBLE_INT_) B
REAL(KIND=_DOUBLE_) C

INTEGER LL
PARAMETER (LL=64)
INTEGER(KIND=LL) D
REAL(KIND=LL) E
```

Typed Constants

Fortran 90 introduces the notion of typed constants. To specify the type (including kind) of a literal constant, append an underscore and then an integer constant expression that evaluates to a valid **KIND** number. For example, a constant of type **REAL(KIND=_DOUBLE_)** could be written either as **1.0D0** or as **1.0__DOUBLE_**.

Note the double underscore when the appended underscore of the typed constant is combined with the prepended underscore of the CM Fortran symbolic name for the kind type parameter:

```
INTEGER LL
PARAMETER (LL=64)
INTEGER(KIND=LL) D
D = 1234_64 + 1234_LL + 1234__DOUBLE_INT_
```

3.2 Enhanced Intrinsic Functions

Another component of the Fortran 90 kind mechanism is the intrinsic inquiry function **KIND**, which takes a scalar or array argument and returns the kind type parameter of the argument. The returned value is itself a scalar of the default integer type (**INTEGER*4** in CM Fortran). For example,

```
INTEGER I
INTEGER*8 J
...
PRINT *, KIND(I)          ! prints 32
PRINT *, KIND(J)          ! prints 64
```

The **KIND** function can be referenced in specification statements as well as in executable statements.

In addition, the intrinsic functions **INT** and **NINT**, which convert a value of any numeric type to an integer, have been enhanced to take an optional **KIND** argument. The **KIND** argument is an initialization expression (constant) that specifies the desired length of the result. Examples of **INT** referenced with the **KIND** argument are:

```
INTEGER*8 BIGINT
INTEGER*4 SMALLINT
...
BIGINT = INT(SMALLINT,KIND=64)
...
BIGINT = INT(SMALLINT,KIND=KIND(BIGINT)) !KIND intrinsic
```

The three elemental intrinsics that inquire about the bit-level representation of integer and logical values behave as follows:

- **POPCNT** and **POPPAR**, which report the population count and population parity of the argument, always return an **INTEGER*4** result, for any argument of type **INTEGER*4**, **INTEGER*8**, or **LOGICAL**.
- **LEADZ**, which reports the number of leading 0 bits before a 1 bit is encountered, returns an **INTEGER*4** result for **INTEGER*4** and **LOGICAL** arguments, but returns **INTEGER*8** for arguments of type **INTEGER*8**.

The new and enhanced intrinsic functions all have on-line man pages. View them with the command **man**, specifying the function name in uppercase.

3.3 Utility Library Support

The 64-bit integer type can be used for any CM array argument to procedures in the CM Fortran Utility Library. It may not be used, however, for integer scalar arguments. For front-end array arguments, such as the array argument to the dynamic allocation utilities, an `INTEGER*8` argument is accepted, but the extra length is not used. When passing a type as an argument to a utility procedure, use the predefined constant `CMF_LONG_S_INTEGER`.

For generating pseudorandom numbers, the Utility Library provides a new procedure specifically for 64-bit integers:

```
CMF_RANDOM_LONG_S_INTEGER (DEST, LIMIT)
```

This utility takes only an `INTEGER*8` argument, and it runs the cellular automaton that generates the values for 64 generations. The `LIMIT` argument is also an `INTEGER*8`. The other random number utility, `CMF_RANDOM`, accepts CM arrays of any numeric type, including `INTEGER*8`, but it runs the automaton for only 32 generations, and it accepts only a 32-bit `LIMIT` argument. Man pages are provided on-line for both utilities.

3.4 Restrictions on 64-Bit Integers

- `INTEGER*8` values cannot be used in `READ`, `WRITE`, and `PRINT` statements as anything other than the I/O list items (the data to be transferred). This type cannot be used for unit numbers, record numbers, status variables, and so on.
- `INTEGER*8` values cannot be used as scalar integer arguments to the CM Fortran Utility Library procedures. This type can be used for front-end array arguments, but the second 32 bits of the argument are ignored.
- As with `INTEGER*4`, the CM system does not detect integer overflow for `INTEGER*8` calculations. For example, multiplying 2**62 by 2**62 gives incorrect results without warning. Further, the partition manager and the vector units produce different incorrect results if overflow occurs while converting an `INTEGER*8` to an `INTEGER*4`.
- If an `INTEGER*8` is used as the index variable in a `DO` loop, the total number of iterations cannot exceed (2**32)-1.