Chapter 1

Introduction

CM Fortran is an implementation for Connection Machine systems of the programming language Fortran 77, extended with array-handling and certain other facilities from the language Fortran 90. CM Fortran supports all features of Fortran 77 that control allocation or access to data residing on the control processor. Some restrictions are imposed on the use of Fortran 77 features that depend on the storage order of data residing in the (distributed) memory of parallel processors. Figure 1 illustrates the relationship of CM Fortran to Fortran 77 and Fortran 90.

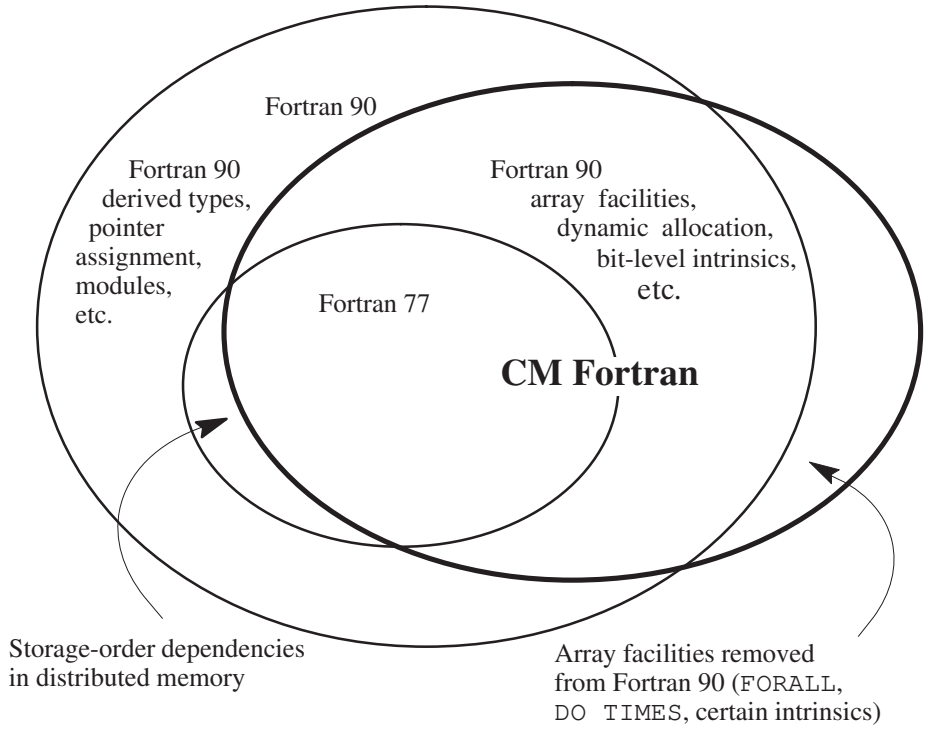

Figure 1. CM Fortran and its relationship to other Fortran dialects.

Copyright © 1989–1994 Thinking Machines Corporation

The following features of CM Fortran have been adopted from the Fortran 90 standard.

- array assignment
- array sections, including sections produced by vector-valued subscripts
- array constructors
- arithmetic and logical operations involving whole array or array sections
- intrinsic functions that perform operations elementally on array or scalar arguments
- intrinsic functions that transform arrays from one shape to another
- keywords in intrinsic function references
- dynamic array allocation with the `ALLOCATE` statement
- array pointers (but not the `TARGET` attribute or pointer assignment statement)
- the `CASE` construct
- the `WHERE` statement and block `WHERE` construct
- array-valued external functions
- interface blocks to specify the interface of external subroutines and functions

CM Fortran also includes the following features from the Fortran 90 standard; these features form part of the DOD military standard extensions (MIL-STD–1753).

- bit-manipulation intrinsic functions
- `DO WHILE` statements
- `END DO` statements
- `INCLUDE` lines
- a `NONE` specifier in the `IMPLICIT` statement

CM Fortran also includes array facilities drawn from earlier versions of the draft standard, but removed from the Fortran 90 standard, including:

- `DO TIMES` statements
- `FORALL` elemental array assignment
- the intrinsic functions `DIAGONAL`, `RANK`, `REPLICATE`, `PROJECT`, `FIRSTLOC`, and `LASTLOC`

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

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

CM Fortran contains some non-compliant syntax drawn from early draft versions of the Fortran 90 standard. This syntax is now supplemented with compliant syntax, both described in this manual. In many cases, both syntax variants are accepted. Where the syntax variants are incompatible, the compiler switch `-[no]f90syntax` controls which is accepted. See the *CM Fortran User's Guide* for a full description of `-[no]f90syntax`.

Copyright © 1989–1994 Thinking Machines Corporation

1.1 Execution Model

A CM Fortran program typically contains several different kinds of source constructions, which the compiler translates into two code streams, serial and parallel. Serial code executes on the control processor; parallel code executes on the parallel processors.

Serial code includes statements that affect the flow of execution within a program, such as `IF`, `DO`, and `CALL` statements, as well as operations that affect only data that resides in the control processor's memory. Scalar data and arrays called *front-end arrays* reside on the control processor. Depending on the execution model chosen at compile time, the control processor may be the partition manager or each of the individual nodes. (See the *CM Fortran Programming Guide* for information on execution models.)

Arrays that are subject to array operations are allocated in the distributed memory of the parallel processing unit, and the array operations are performed by the parallel processors. For historical reasons, CM Fortran calls distributed arrays *CM arrays.* The terms *parallel processor* and *processing elements* refer to the CM system components that perform parallel operations. Depending on the execution model chosen, they may be the *nodes* or the *vector units* under the control of the partition manager, *or* they may be the four vector units of each node under the control of the microprocessor of that node.

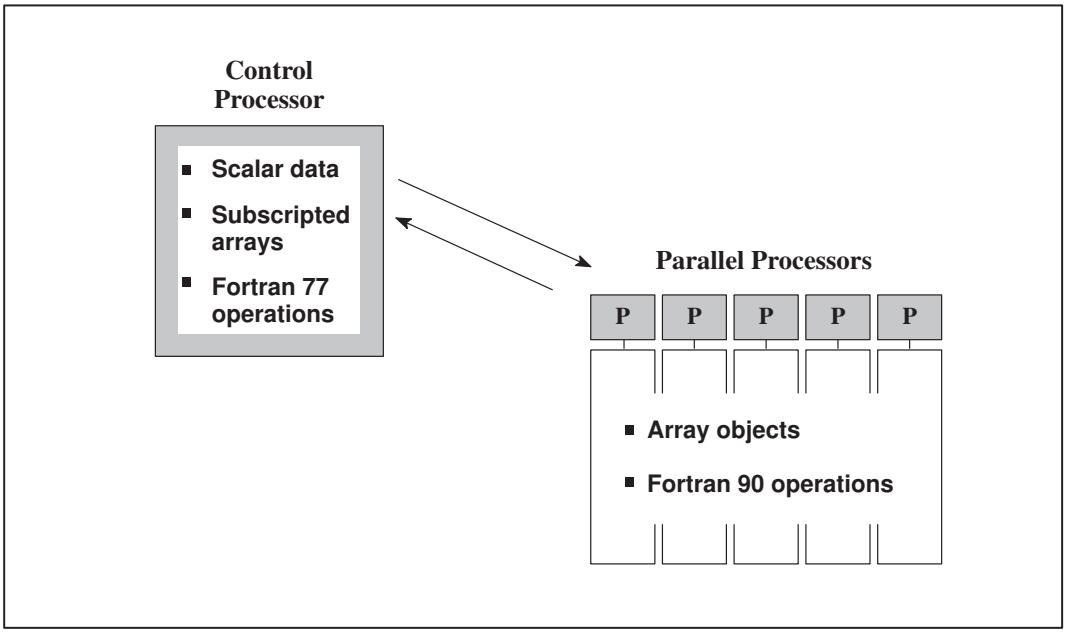

Figure 2. Division of labor between the control processor and the parallel processors.

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

The CM system software dynamically allocates one or many array elements in the memory of each parallel processor, thus making application code independent of the number of physical processors in the machine on which it is executing. Such code *scales* from one size machine to another without recoding or recompilation.

System software handles the distribution of array elements among parallel processors transparently to the user, although the user can influence or even control array layout by means of compiler directives.

The fastest mechanism for operating on or communicating among array elements is local memory accesses within each of the parallel processors. Careful attention to array layout (using the compiler directive `LAYOUT`) can maximize the use of this mechanism.

To transfer data among processors, the CM system software supports three *interprocessor communication* mechanisms, implemented as *run-time communication routines.* The compiler invokes the appropriate mechanism transparently to the user.

- *General communication.* The most general mechanism allows data to be sent from any processor directly to any other processor; many processors can send data to many other processors simultaneously. This mechanism is the most expensive in terms of performance.

- *Grid communication.* A more efficient mechanism treats data as an *n*-dimensional grid and allows every processor to send data to its immediate neighbors in the grid. This grid communication is also called NEWS communication, for historical reasons.

- *Global communication.* A third mechanism performs cumulative operations over an array or along a dimension. The cumulative operation may reduce an array or a dimension to a scalar result, or it may *scan* a dimension to accumulate a set of results in the elements on that dimension.

Array assignment in CM Fortran is often performed without the need for any interprocessor communication, but it may in some cases require grid or general communication. Vector-valued subscripting in an array section requires the general communication mechanism, as do transformational intrinsic functions such as `PACK` and `UNPACK`. The intrinsic function `CSHIFT` usually uses grid communication when performing a constant shift along an array dimension. References to intrinsic reduction functions such as `SUM` use the global communication mechanism.

Copyright © 1989–1994 Thinking Machines Corporation