Chapter 4

CM-5 Languages and Libraries

This chapter gives an overview of the languages, libraries, and related software currently available for the CM-5. For complete information, see the manuals for the individual products.

4.1 CM Fortran

Fortran for the Connection Machine system is standard Fortran 77, supplemented with the array-processing extensions of Fortran 90. These features, well known in the Fortran community, are particularly useful in data parallel programming.

The array-processing extensions provide convenient syntax and numerous intrinsic functions for manipulating arrays. The Fortran 90 syntax allows an array to be treated as a first-class object. Thus, in the statement **`A = A + 1`**, the variable **`A`** can be a scalar, a vector, a matrix, or a higher-dimensioned array. In any case, the statement causes all elements of **`A`** to be operated on in parallel. The syntax also permits specifying sections of arrays, which can be used anywhere whole arrays are used: in expressions and assignments and as arguments to procedures.

CM Fortran also supports Fortran 90 dynamic array allocation, 64-bit integers, and the **`FORALL`** statement for location-dependent assignments of array elements, all in parallel. The standard **`READ`** and **`WRITE`** statements transfer data in parallel, that is, in multiple streams, when the transfer is between parallel memory and a CM storage device

Copyright © 1993–1994 Thinking Machines Corporation

Existing Fortran 77 programs can be converted into CM Fortran with the assistance of the CMAX Converter, discussed in Section 4.6. This tool automatically translates **DO** loops on arrays into array syntax.

4.1.1 Programming Models

Version 2.1 of CM Fortran on the CM-5 supports both global and local views of program execution. (Earlier versions support only the global execution model.)

- When used alone, CM Fortran operates in a global manner. That is, it lays out parallel arrays across the processing nodes (or, optionally, across the vector units) of the CM-5 partition on which it executes. Each processing element (node or vector unit) computes on its local data, independently of the others. The partition manager executes scalar code and calls run-time functions for interprocessor communications. Race conditions cannot develop because the processing elements are synchronized after every block of parallel computation and after every communication operation.

- When used together with the message-passing library CMMD, CM Fortran operates in a local manner. Copies of the CM Fortran program run independently on each node, each using its associated four vector units to perform array operation sin parallel. Scalar data and serial operations are handled by the microprocessor on the node. the programmer handles communication and synchronization among the programs explicitly, using CMMD message-passing routines.

- CM-5 systems also support a hybrid, "global/local" mode of execution, which enables global programs to take advantage of message-passing techniques. A global/local program begins with a global main program executing in data parallel fashion. At any time thereafter, the application can take explicit control of the nodes by calling a local routine. Invoking the local routine temporarily transforms the application into a nodal program, executing in message-passing style. A copy of the local routine runs on each node, using the four associated VUs as its parallel processors.

Copyright © 1993–1994 Thinking Machines Corporation

4.1.2 Intrinsic Functions

CM Fortran extends all the Fortran 77 intrinsic functions (except character functions) such that they can take either a scalar or an array argument. For arrays, they apply the operation to all the elements. Fortran 90 also defines a rich set of intrinsic functions that operate only on arrays. Some, such as **CSHIFT** and **TRANSPOSE**, reposition the elements. Others, such as **SUM** and **MAXVAL**, combine array elements, reducing them to a scalar or an array of lower rank. Still other transform arrays, like **RESHAPE**, **PACK**, and **SPREAD**, or multiply arrays, like **DOTPRODUCT** and **MATMUL**. In addition, CM Fortran offers bit-level intrinsics, like **IAND** and **ISHIFT**, and inquiry functions like **FIRSTLOC**, **LEADZ**, **POPCNT**, and **POPPAR**.

4.1.3 Utility Library

The CM Fortran Utility Library provides a convenient interface to CM-5 operations that the language cannot express easily or that the compiler does not yet generate. The Utility Library provides such capabilities as ranking and sorting, array and system inquiry, and table look-up. The "aliasing" utilities provide a form of "equivalence" (or storage association) on the local subgrids of a distributed array.

4.1.4 Development and Monitoring Facilities

CM Fortran programs are typically developed within the Prism programming environment, with its graphical debugger, performance analysis tools, and data visualizers.

Timing of CM Fortran programs is accomplished using the CMOST timing functions, such as **CM_timer_start**, **CM_timer_print**, and so on.

Run-time safety, including array argument checking and some NaN checking, is enabled by the **cmf** compiler switch **-safety**.

4.1.5 Documentation Provided

- *Getting Started in CM Fortran*

Copyright © 1993–1994 Thinking Machines Corporation

- *CM Fortran Programming Guide*
- *CM Fortran Language Reference Manual*
- *CM Fortran Libraries Reference Manual*
- *CM Fortran User's Guide*
- *CM-5 CM Fortran Performance Guide*
- *CM Fortran Array Operations Quick Reference Guide*
- On-line manual pages for all CM Fortran intrinsic functions, the **`cmf`** command, and library procedures (utility, global/local, and cmf77 libraries), and release notes for the version of CM Fortran you are using.

4.2 C*

C* is an extension of the C programming language, designed to support data parallel programming. It is based on the standard version of C specified by the American National Standards Institute (ANSI).

C* extends C with a small set of new features that allow programmers to use the Connection Machine system efficiently. Examples are **`shapes`**, which are used to logically configure parallel data; parallel versions of arrays and structures; and **`where`** statements, which restrict the set of positions within an array on which operations are to take place.

C* also adds a few new operators to Standard C. For example, the **`<?`** and **`>?`** operators are available to obtain the minimum and maximum of two variables (either scalar or parallel). The corresponding compound assignment operators **`<?=`** and **`>?=`** are also provided. The operator **`%%`** provides a true modulus operation (as compared to the remainder operator **`%`**).

In addition, C* provides parallel functions for computation and communication. Functions may be overloaded: you can declare more than one version of a function with the same name (one for scalar data, for example, and another for parallel data). The compiler automatically chooses the right version.

C* offers interfaces to CMMD, CMX11, CM/AVS, and CMSSL (as of CMSSL Version 3.2). It provides I/O calls that can access all CM-5 file sys-

Copyright © 1993–1994 Thinking Machines Corporation

tems, including SFS (for the SDA) and CMFS (for the DataVault and other CM-2/200 devices); as of Version 7.1.1 it also provides support for HIPPI and UNIX sockets.

4.2.1 Programming Models

Like CM Fortran, C* supports both data parallel and message-passing programming styles.

- When compiled with the **`-sparc`** or **`-vu`** option, a C* program operates in a data parallel manner. It handles scalar data on the partition manager, and lays out all parallel data across all the processing nodes of the CM-5 partition on which it executes. The **`-vu`** option makes use of the optional vector units on the CM-5; the **`-sparc`** option ignores the vector units.

- When used with the **`-node`** option, the C* program operates in a message-passing manner. Independent copies of the C* program run on every node. Scalar data is handled by the node's microprocessor; parallel data is handled by the vector units. Communication among processors is handled by calls to the CMMD library.

4.2.2 Documentation Provided

- *Getting Started in C**
- *C* Programming Guide*
- *CM-5 C* User's Guide*
- *CM-5 C* Performance Guide*
- Release notes for the version of C* that you are using

4.3 CM Scientific Software Library

The Connection Machine Scientific Software Library (CMSSL) is a growing set of numerical routines that support computational applications while exploiting the massive parallelism of the Connection Machine system.

Copyright © 1993–1994 Thinking Machines Corporation

CMSSL provides data parallel implementations of familiar numerical routines in the areas of linear algebra, ordinary differential equations, signal processing, statistical analysis, and linear programming. It also offers a number of communication functions that facilitate computations on both structured and unstructured grids.

On the CM-5, CMSSL is callable from CM Fortran; most operations are also available from C*. The user interface for the CM Fortran routines is identical with the CM Fortran user interface offered on the CM-200; the actual implementation of the routines, however, often differs, to take best advantage of the hardware of each machine.

Version 3.2 of CMSSL concentrates on six critical areas of programming: numerical linear algebra, Fourier Transforms, ordinary differential equations, linear programming, random number generation, and statistical analysis.

4.3.1 Linear Algebra

Most CMSSL linear algebra routines are designed to support multiple instances. The difference between invoking computation on a single instance and on multiple instances lies only in the dimensionality and layout of the data structures used as parameters to the CMSSL routine.

Within the general area of linear algebra, CMSSL offers

- Matrix operations on dense, grid sparse, and arbitrary sparse matrices. For dense matrices, CMSSL includes inner and outer product routines; matrix, matrix vector, and vector matrix multiplication routines; a 2-norm routine; and an infinity norm routine. For grid and arbitrary sparse matrices, the library provides matrix, matrix vector, and vector matrix multiplication.

- Linear equation solvers for dense, banded, and sparse systems of equations: *LU* and *QR* factorization and solution routines (both in-core and out-of-core), triangular system solvers, a Gauss-Jordan system solver, and matrix inversion; factorization and solution of banded systems via pipelined Gaussian elimination (with optional pairwise pivoting) or via substructuring with either cyclic reduction, balanced cyclic reduction, pipelined Gaussian elimination, or transpose; and several standard iterative solvers, including the conjugate

Copyright © 1993–1994 Thinking Machines Corporation

gradient, bi-conjugate gradient with stabilization, Quasi-Minimal Residual (QMR), and restarted Generalized Minimal Residual (GMRES) methods.

- Eigensystem analysis of real symmetric tridiagonal, dense Hermitian, dense real symmetric, dense real, and sparse systems, via a number of methods including Jacobi rotations, *k*-step Lanczos method, and *k*-step Arnoldi method.
- Singular value decomposition of dense real matrices.

4.3.2 FFTs

CMSSL offers routines for the computation of Fast Fourier Transforms (FFTs) by Cooley-Tukey type algorithms on one or more axes of arrays with an arbitrary number of axes. The complex-to-complex FFT is a Discrete Fourier Transform in which the same call is used to specify a one-dimensional or multidimensional transform on any legal CM array or parallel variable of floating-point data. The real-to-complex FFT computes the Fourier Transform of real data; the complex-to-real FFT transforms conjugate symmetric sequences.

4.3.3 Ordinary Differential Equations

CMSSL provides a routine that solves the initial value problem for a system of *N*-coupled first-order ordinary differential equations by explicitly integrating the set of equations using a fifth-order Runge-Kutta-Fehlberg formula.

4.3.4 Linear Programming

CMSSL provides a routine that solves multidimensional minimization problems using the simplex linear programming method. The goal is to find the minimum of a linear function of multiple independent variables.

Copyright © 1993–1994 Thinking Machines Corporation

4.3.5 Random Number Generation

CMSSL provides two random number generators, the *Fast RNG* and the *VP RNG*. Both use a lagged-Fibonacci algorithm to produce a uniform distribution of random values. Both may be reinitialized for reproducible results. The Fast RNG uses less processing-node memory; the VP RNG can produce identical results on partitions of different sizes.

4.3.6 Statistical Analysis

CMSSL offers two histogramming operations: one that tallies the occurrences of each value in a CM array, and one that counts the occurrences of values within specified value ranges. The latter, the *range histogram*, facilitates breaking data from particularly large data sets into subranges, perhaps as a preliminary step before doing more detailed analysis of interesting areas. Two versions of the range histogram are provided; one stores the histogram on the partition manager, and the other stores it on the processing nodes.

4.3.7 Communication Functions

CMSSL includes routines for efficient data motion for nearest-neighbor operations on regular grids, for all-to-all communication on segmented arrays, and for gather and scatter operations on unstructured grids. The library also provides utilities for data distribution for load balancing of communication. Routines offered include polyshift; all-to-all rotation, broadcast, and reduction; transpose; several gather and scatter utilities; and partitioning of an unstructured mesh. There is also a communication compiler, a set of routines that compute and use message delivery optimizations for basic data motion and combining operations. The communication compiler allows you to compute an optimization just once, and then use it many times in subsequent data motion and combining operations.

Copyright © 1993–1994 Thinking Machines Corporation

4.4 Visualization Programming

4.4.1 A Distributed Graphics Strategy

In keeping with its role as a network resource, the CM-5 uses a distributed graphics strategy to support a wide range of user applications. The key items in this strategy are

- the parallel processing power of the Connection Machine supercomputer
- the specialized power and interactive visualization environments, such as AVS, provided by dedicated graphics display stations
- the use of standard protocols, such as X11, to allow communication among a variety of hardware and software

A full range of interconnections is supported, from high-speed HIPPI interfaces through FDDI and Ethernet for longer-distance communications, to allow fast communication between the CM and graphics display stations.

As an example, a scientific visualization program could use the CM to compute image geometry (including, for example, polygon coordinates and color information) and then send it from the CM directly to local memory on the graphics workstation, where the results of simulations done on the CM can be displayed and analyzed interactively.

4.4.2 An Integrated Environment

By using the distributed graphics strategy described above, together with an underlying protocol such as X11 or the AVS graphical user interface, programmers can create and use a wide variety of integrated environments for their computational and visualization tasks. Connection Machine software provides an environment that permits the exchange of very large data sets between the CM and frame buffers, workstations, or X window terminals.

4.4.3 The CM/AVS Visualization Environment

CM/AVS adapts and extends the Application Visualization System (AVS, from Advanced Visual Systems Inc.) to the realm of the CM-5. This graphi-

Copyright © 1993–1994 Thinking Machines Corporation

cal user interface enables an application to operate on data that is distributed on CM-5 processing nodes and to interoperate with data from other sources. A user normally runs AVS on a local workstation and uses the modules and functions that CM/AVS provides to process data on the CM-5. That way, the advantages of user-interface-intensive workstation visualization are combined with the power of data-intensive CM-5 applications.

The building blocks of an AVS application program are small, packaged units of code, called *modules*. Most modules process a set of inputs into a set of outputs. Each module incorporates a function, which can be as simple as adding two arrays or as complicated as rendering the isosurfaces of a volume. AVS modules execute on the workstation; CM/AVS modules execute on the CM-5. Hundreds of visualization modules are available from AVS and Thinking Machines and in the public domain.

Data for module inputs and outputs is typed. CM/AVS provides a parallel version of the AVS "field" data type used to represent arbitrary arrays of data. CM/AVS's parallel field data is allocated on the CM-5 processing nodes as CM Fortran arrays or C* parallel variables.

Within CM/AVS, parallel fields appear identical to regular serial fields; the two may be used interchangeably. When CM/AVS modules that operate on parallel data are connected with AVS modules that operate on serial data, CM/AVS routines convert the data between parallel and serial fields as required. The conversion is transparent to the user and to the module writer.

Documentation Provided

- *CM/AVS User's Guide*
- *CM/AVS Release Notes*
- On-line man pages viewable through the AVS man page viewer, and release notes for the version of CM/AVS you are using

In addition, users should have the AVS document set.

4.4.4 Visualization Programming with CMX11

The CMX11 visualization library is designed for distributed graphics programming in a heterogeneous computing environment. This library, callable

Copyright © 1993–1994 Thinking Machines Corporation

from CM Fortran and C*, allows you to display data from CM-5 memory on an X Window System server screen anywhere on your network.

Creating and Controlling a Display

The CMX11 library provides functional CM Fortran and C* interfaces that make it easy to create and control one or more windows on an X11 server as a CM display.

- Specify the X11 server you wish to use by setting the environment variable **`DISPLAY`** or by using the **`-display`** option on the command line when you invoke your program.
- Within your program, call the subroutine **`CMXCreateSimpleDisplay`** to connect to the display specified.

Other routines allow you to manage the display from your application. For example, you can get information on the display size or set the display colors.

Rendering Your Data

The CMX11 library provides parallel extensions to the standard X drawing primitives that accept parallel arrays of coordinate and color information. These routines enable you to draw large sets of points, lines, arcs, rectangles, filled polygons, text strings, or an image array — each with a single subroutine call.

Graphics Programming

The basic CMX11 drawing and display capabilities do not require any X programming. However, the library provides routines that give you access to the underlying X structures. If you are an experienced X programmer, these enable you to integrate your CMX11 program with an existing X or Motif application.

Copyright © 1993–1994 Thinking Machines Corporation

Documentation Provided

Anyone who wants to perform simple visualization operations on the results of CM-5 computations will find sufficient information in the Thinking Machines CMX11 documentation:

- *CMX11 Reference Manual*
- *CMX11 Release Notes*
- On-line man pages for all CMX11 routines, and release notes for the version of CMX11 you are using

For more elaborate graphics programming, users who are unfamiliar with X may wish to consult these publications in addition:

- *Xlib Programming Manual,* Adrian Nye (Sebastopol, CA: O'Reilly, 1988)
- *Xlib Reference Manual,* Adrian Nye (Sebastopol, CA: O'Reilly, 1990)
- *X Window System User's Guide,* Valerie Quercia and Tim O'Reilly (Sebastopol, CA: O'Reilly, 1990)

4.5 Message Passing with CMMD

The CMMD communications library supports message-passing programming on the CM-5. This programming involves explicit message passing between processing nodes.

The CMMD library is callable from C*, CM Fortran, C, C++, and FORTRAN 77. C* programs are compiled with the **`cs`** compiler, while CM Fortran programs are compiled with the **`cmf`** compiler. C, C++, and FORTRAN 77 programs are compiled with the appropriate Sun compiler. Programs or program modules written in CM Fortran or C* use the vector units on the CM-5, as do modules written in DPEAC or CDPEAC. Other programs or modules use only the microprocessor on each node.

Copyright © 1993–1994 Thinking Machines Corporation

4.5.1 Programming Models

CMMD supports three programming models:

- The host/node programming model involves writing two programs that will run simultaneously. One program runs on the host, while an independent copy of a second program runs on each processing node. On the CM-5, the host is the partition manager that controls a given partition, while the nodes are the processors within the partition. The host begins execution by performing needed initializations (including initializing the CMMD message-passing environment) and then invoking the node program; it may have little involvement in subsequent computations.

- The hostless programming model uses the host only to initiate execution and to act as an I/O server. A CMMD-supplied host program performs these tasks; the user writes a single application, which runs on each of the nodes. The nodes pass messages to each other, but do not explicitly talk to the host.

- The global/local model is currently usable only by CM Fortran programs. In this model, the main program is a global CM Fortran program. Because it is a global program, the host keeps the nodes synchronized and also executes any serial code in the program. Parallel code runs on the VUs, with parallel arrays being laid out by the CM run-time system across all VUs in the partition.

  Within this global program, individual subroutines may be written in *local* mode. While these routines execute, all control occurs at the node level, just as in any message-passing program. Each node treats its subgrid of a global parallel array as though it were a complete array, and the nodes communicate and synchronize with each other via CMMD function calls.

4.5.2 Cooperative Processing and Asynchronous Processing

CMMD supports both cooperative and asynchronous message passing. With cooperative concurrent message passing, synchronization occurs only between matched sending and receiving nodes and only during the act of communication. At all other times, computing on each node proceeds asynchronously with respect to the other nodes.

Copyright © 1993–1994 Thinking Machines Corporation

A set of global functions provides for broadcast, reduce, scan, and concatenate operations, and for global synchronization. As their name implies, global functions involve all nodes in the partition; executing the function synchronizes the nodes.

CMMD also permits fully asynchronous message passing. Asynchronous message passing is usually interrupt-driven: a node signals a readiness to send or receive a message, then performs other work until its partner node is ready for the transmission. If preferred, however, asynchronous message passing may be driven by polling.

To optimize performance of repeated patterns of message passing, CMMD provides virtual channels. With channels, two nodes establish a one-way transmission link that can be used multiple times. Once the channel is established, the sending node writes a predefined array into the channel; the receiving node reads the channel, then resets it for another use. No synchronization is needed.

4.5.3 Active Messages and Block Transfers

In addition to the message-passing functionality listed above, which uses a handshake protocol, CMMD provides protocol-free messaging capabilities. These are particularly useful for programmers who want to define their own protocols. Two major items in this area are the Active Message facility and block data transfers.

- With the Active Message facility, one node activates some routine on a second node. The routine may itself activate further routines that either respond to the first node or activate routines on yet other nodes.

- With block data transfers, one or more nodes write data to a "receive port" on a destination node. The nodes agree in advance on the size and nature of the transfer, so that the receiving node can set up the port correctly prior to the transfer. Senders and/or receivers can specify handler functions to be invoked automatically when the transfer completes.

Copyright © 1993–1994 Thinking Machines Corporation

4.5.4 CMMD I/O

CMMD extends UNIX I/O to provide for both independent and cooperative I/O. A file may be open for a single node or for all nodes. If open for all nodes, it can be in one of three modes:

- In independent mode, any node can read or write the file independently.
- In synchronous-sequential mode, all nodes read and write the file simultaneously, each reading or writing a separate, but sequential, portion of the file.
- In synchronous-broadcast mode, one portion of a file is read and broadcast to all nodes simultaneously.

CMMD also provides double-precision file pointers, to allow applications to access very large files.

4.5.5 Supporting Utilities

The supporting facilities provided for data parallel programs, such as the CM timers, typically treat the nodes as a collective, since the nodes each store part of the same data set. Message-passing programs, in contrast, are supported by facilities that allow independent access to each node: for example, the CMMD timers and Node Prism. The *CMMD User's Guide* provides hints for using program development and monitoring facilities. The *Prism User's Guide* describes how to use Node Prism to debug a message-passing program.

4.5.6 Documentation Provided

- *CMMD Reference Manual*
- *CMMD User's Guide*
- On-line man pages for all CMMD routines, and release notes for the version of CMMD you are using

Copyright © 1993–1994 Thinking Machines Corporation

4.6 The CMAX Converter

CMAX — the "CM Automated X-lator" — is an aid to converting standard FORTRAN 77 into CM Fortran. CMAX provides a convenient migration path for serial programs onto the massively parallel Connection Machine system, both for data parallel applications and for CM Fortran/CMMD message-passing applications.

In addition, CMAX gives users the option of maintaining their software in FORTRAN 77 for maximum portability to multiple platforms. Users in a heterogeneous computer environment and third-party software developers can use the converter as a "preprocessor" for routine Fortran compilation for CM systems. In this sense, CMAX provides a migration path onto and off of the Connection Machine system.

The major difference between serial and data parallel Fortran programs is the substitution of array operations for loop iterations, and the concomitant need to lay out some arrays across the processing nodes. These are the tasks performed by the CMAX converter.

CMAX is a **DO** loop vectorizer. It analyzes loop constructs and translates them into CM Fortran array operations. For greatest efficacy, the converter performs an interprocedural dependence analysis of the whole program (not just of individual subroutines) and applies vectorization techniques such as loop fissioning, scalar promotion, and loop pushing to the input code. CMAX also recognizes the intent of numerous programming idioms, such as structured data interactions and dynamic array allocation. When translating code, it makes full use of powerful Fortran 90 features such as array-processing intrinsic functions and dynamic allocation statements, as well as the **FORALL** statement defined by High Performance Fortran. CMAX thus provides entree both to the Connection Machine system and to the emerging HPF standard.

CMAX gives the user the option of translating all or part of a FORTRAN 77 application to CM Fortran. By using the **firewalls** option, you can concentrate the conversion effort on the computational kernel of a program — the part that usually benefits most from parallel processing — while leaving setup and I/O operations unchanged. CMAX generates code to guarantee that the converted and unconverted portions of the program can be linked into a single executable to run on the Connection Machine system.

CMAX provides a convenient interface to the user. The Prism development environment provides facilities for examining CMAX output and comparing

Copyright © 1993–1994 Thinking Machines Corporation

it line-by-line with the input program; see Figure 16, which shows a Prism window with the CM Fortran code in one pane, and the corresponding FORTRAN 77 code in the other pane. CMAX command options and in-line directives allow the user to control the converter's actions and decision rules. The CMAX library provides canonical, portable — and translatable — FORTRAN 77 utilities for expressing common operations like dynamic array allocation and circular array element shifts. The converter generates detailed notes of a conversion, explaining all the changes it has made.

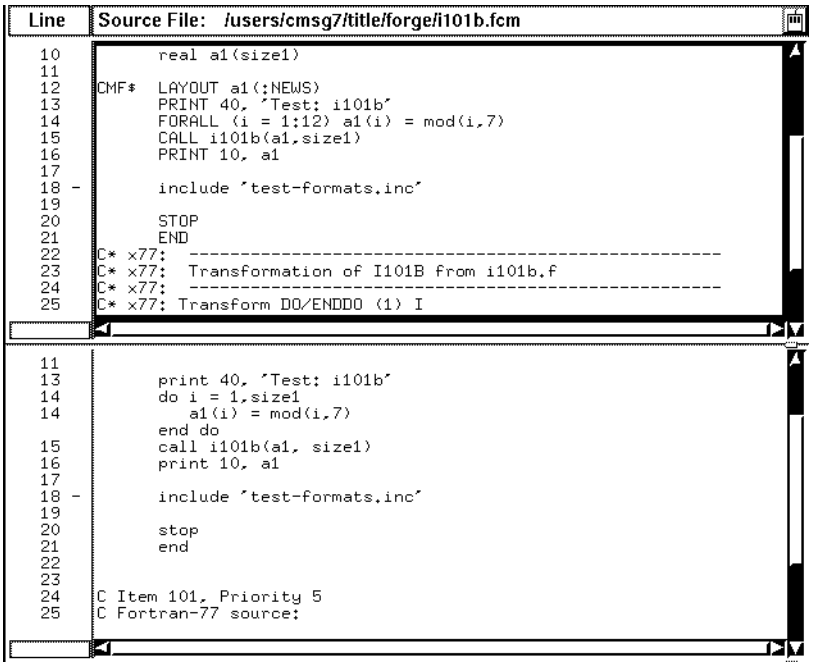

Figure 16. CM Fortran and FORTRAN 77 code in a split screen.

Although CMAX is designed primarily to assist in the creation of new applications, it accepts as input any program that is written in standard FORTRAN 77 and follows standard guidelines for scalability. These simple guidelines guarantee that a program runs efficiently on any size data set, large or small, and on any number of processors, from one to thousands. The combination of guidelines plus converter can assist substantially the task of upgrading "dusty deck" programs to take advantage of modern architectures and language features.

Copyright © 1993–1994 Thinking Machines Corporation

The conventions of scalable Fortran programming express three basic objectives:

- Make it easy for a compiler to recognize how data and computations may be split up for independent or coordinated processing. For example: loop over as many array axes as possible in a single operation; use standard idioms to express common, well-structured data dependences.
- Avoid constructions that rely on a particular memory organization, such as linearizing multidimensional arrays or changing array size or shape across program boundaries.
- Use data layout directives and library procedures (with some conditionalizing conventions) to take advantage of the specific performance characteristics of each target platform. For example, FORTRAN 77 programs targeted to the CM system can use compiler directives to fine-tune data layout and access the CM libraries for procedures that are specially tuned for performance on the CM system.

4.6.1 Documentation Provided

- *Using the CMAX Converter*

4.7 Assembly Language

The CM-5's processing node currently uses a SPARC microprocessor. Low-level programming can therefore be carried out in SPARC assembly language; this is not recommended, however, since the implementation of the CM-5 processing node is expected to track the RISC microprocessor technology curve to provide the best possible functionality and performance at any given time. Thus, SPARC assembly language programs may not be portable to future CM-5s.

Thinking Machines provides an instruction set called DPEAC for programming the optional vector-unit hardware. DPEAC is an extension of SPARC assembly language, providing additional assembly-level operations that are CM-5 specific.

Copyright © 1993–1994 Thinking Machines Corporation

4.7.1 Documentation Provided

- *DPEAC Reference Manual*

Copyright © 1993–1994 Thinking Machines Corporation