The
Connection Machine
System

CM-5 Software Summary

CMOST Version 7.2 Beta
March 1993

Thinking Machines Corporation
Cambridge, Massachusetts

First beta printing, March 1993

Thinking Machines Corporation
245 First Street
Cambridge, Massachusetts 02142-1264
(617) 234-1000/876-1111

CM-5 Software Summary

1 Parallel Programming on the CM-5

The Connection Machine CM-5 supercomputer offers users high performance through a complete range of parallel processing approaches. The CM-5 supports both data parallel programming and message-passing programming:

- The CM's data parallel compilers (CM Fortran, C*, *Lisp) present the user with a global address space and a single thread of control.
- The CMMD communications library, callable from Fortran 77, C, and C++ (as well as from CM Fortran and C*), provides fast communication between independent tasks (or threads of control).
- Both models utilize the rapid synchronization and low-latency communication capability of the CM-5.

In the past, programmers of supercomputers were forced to choose between these two models. The CM-5, however, supports both; in fact, data parallel programs and message-passing programs can run simultaneously, under timesharing, on a single partition of a CM-5.

This document summarizes the CM-5's operating system (CMOST, Version 7.2) and the layered software products it supports.

1.1 Data Parallel Languages

Programs written in data parallel languages take a global view of the system. In this model, a single program executes on the control processor. This program controls all the processing nodes, requesting synchronized computation, communication, and I/O as needed. These programs use Connection Machine data parallel compilers and run-time system to control data layout and inter-processor communication and synchronization. They also make use of the CM's specialized parallel library routines and I/O functionality.

With data parallel programming, the programming nodes work in synchrony. For example, let's consider a finite difference code that needs to perform one operation on its boundary elements and another on interior elements. Code written to deal with this case in a data parallel language would look something like

```
where (boundary_elements)
   do_a
elsewhere
   do_b
end where
```

This single flow of control, so similar to that of a standard "serial" program, makes data parallel programs the easiest of parallel programs to debug, and helps account for the popularity of this programming style.

For more information on data parallel programming, we recommend that you consult the CM Fortran manuals (especially *Getting Started in CM Fortran* and the *CM Fortran Programming Guide*).

1.2 Message-Passing Programming

Message-passing programs take a node-level view of the system. Again, a single program executes; but in this programming style, a separate copy of the program executes independently on each node. The nodes divide tasks and data among themselves according to the needs of the application; they may stay closely synchronized or become completely asynchronous. All communications and synchronization, as well as data layout, are under the application's explicit control.

The message-passing programming style is most useful when an application requires the dynamic allocation of tasks or data. Such applications typically use a class of algorithms known as node-expansion algorithms: examples are divide and conquer algorithms, branch and bound algorithms, asymmetric traveling salesman problems, and tree search problems.

On the CM-5, message-passing programs utilize the CM's communications library, CMMD. CMMD functions can be called from C, C++, Fortran 77, CM Fortran, or C*; the ability to use standard high-level languages is appreciated by users who have existing programs that they wish to port to a parallel supercomputer.

1.3 Scalable Computing

The ability to choose among programming models is an important feature of the CM-5, as it lets users choose the technique that is best, not only for their application, but for each part of their application. Also important is the Connection Machine's support for Scalable Computing, and its provision of tools geared specifically to the needs of its users, such as the Prism programming environment and the CMAX Fortran 77-to-CM Fortran translator.

Connection Machine data parallel programming has always been inherently scalable. Because the CM's data parallel software lays out data arrays at run time, a single program can run on any size Connection Machine, with computation and communication patterns optimized for machine size. Now, the CM-5 allows message-passing to be scalable as well.

1.4 CM-5 Software

As suggested above, the Connection Machine CM-5 provides software to support both data parallel and message-passing programs. Figure 1 diagrams the current CM-5 software offerings.

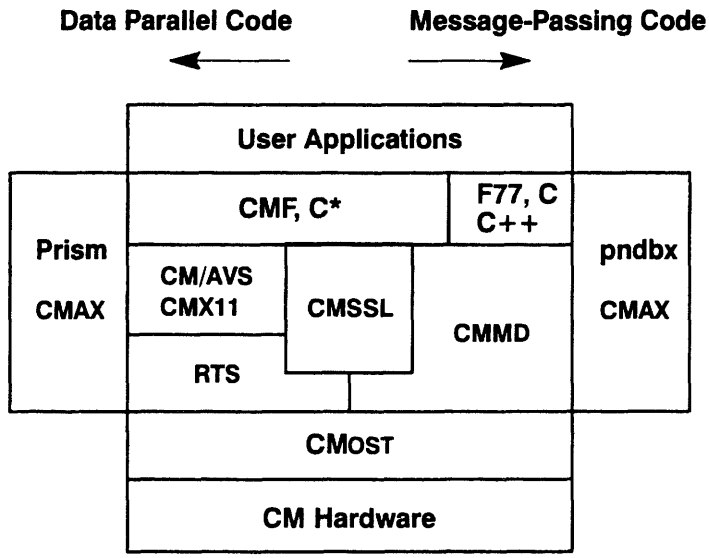

Figure 1. CM-5 software.

1.4.1 Data Parallel Software

- The CM-5 currently offers three high-level languages for data parallel programming: CM Fortran, C*, and *Lisp. These languages are nearly identical to their CM-200 counterparts. A few implementation-specific differences exist: these are detailed in Section 4 for CM Fortran, and Section 5 for C*. Readers interested in *Lisp should consult *Porting to CM-5 *Lisp.*

- Data parallel libraries at this release are CMSSL (the CM Scientific Software Library) and CMX11 (a visualization library that provides parallel extensions to the X11 standard. Section 6 briefly introduces CMSSL; Section 7 introduces CMX11.

- CM/AVS is a graphical user interface for visualization that links the computing power of the CM-5 with the convenience of a specialized graphics workstation. It is described in Section 7.

1.4.2 Message-Passing Software

Message-passing programs on the CM-5 can be written in C, C++, Fortran 77, CM Fortran, or C*. They are supported by a message-passing library, CMMD. This library and its associated support utilities in the operating system are summarized in Section 8 of this overview.

1.4.3 Assemblers

The assembly language for the CM-5 is DPEAC; its assembler is **dpas**. The CM Fortran and C* compilers compile programs into DPEAC code.

The DPEAC language is a superset of the SPARC **as** assembly language. Sun's C, C++, and Fortran 77 compilers compile programs into **as**.

On a CM-5, **as** statements can control the SPARC microprocessors in the nodes and in the partition manager; DPEAC statements can control both the SPARC and the vector units.

The **dpas** assembler reads DPEAC sources and outputs UNIX object (`.o`) files. It does this by preprocessing its source, translating the DPEAC-specific statements into SPARC assembly source, and running the resulting text through the **as** assembler. Any SPARC assembly statements are passed unchanged to the **as** assembler.

Because the **dpas** assembler contains an excellent preprocessor for both **as** and DPEAC statements, it is sometimes very useful as a preprocessor for an **as** program.

For full information on DPEAC and **dpas**, see the *DPEAC Reference Manual.*

1.4.4 CMAX

CMAX — the "CM Automated X-lator" — assists the conversion of standard Fortran 77 into CM Fortran. CMAX provides a convenient migration path for serial programs onto the CM-5. Software may be maintained in either Fortran 77 or CM Fortran. CMAX is described in Section 9.

1.4.5 The CMOST Timeshared Operating System

The CM-5 operating system, CMOST, is an enhanced version of the UNIX operating system. As such, it enables the CM-5 to interact efficiently with other devices in a heterogeneous, networked computing environment, while at the same time managing the interactions of CM-5 system components and managing the timeshared execution of multiple parallel programs.

Section 2 gives a brief overview of the CMOST operating system.

1.4.6 Prism Programming Environment

Prism is a programming environment that integrates debugging, profiling, and other useful tools in a convenient windowed environment. A graphical interface is available from terminals and workstations that are running the X Window System. A command interface is available from other terminals. Although designed primarily for use with data parallel programs, Prism can also be useful for other program development. Prism is described briefly in Section 3 of this document.

2 The CMOST Operating System

The CMOST operating system provides all the capabilities of UNIX, along with special services for parallel programmers of the Connection Machine CM-5.

As a resource on a heterogeneous computing network, CMOST provides:

- standard UNIX utilities, user interfaces, protection, and security
- support for all standard UNIX-based communications protocols
- the ability to access files and exchange data with other systems

As a resource within the CM-5 system, CMOST provides:

- fast interprocessor communications
- the parallel operations required for best utilization of processing hardware, especially the array of processing nodes
- central administration and resource management to support all CM-5 computation and I/O facilities
- timeshared execution of parallel programs

2.1 Configuring the CM-5

The CM-5 is a flexibly partitionable system, with partitioning under the control of the system administrator. A partition consists of a set of processing nodes (PNs) and the control processor (CP) that controls them. Each partition operates independently of the others.

In its role as partition manager, the control processor runs a full version of the CMOST operating system. Once logged in to the partition, users have full access to the control processor itself, to all the nodes in the partition, and to all the CM-5's I/O resources and network connections. The system administrator determines the degree of access permitted to each partition, using standard UNIX access control mechanisms.

Information on system administration for the CM-5 is given in *CM-5 System Administrator's Guide,* Version 7.2.

2.2 Executing Parallel Programs

Since CMOST is an enhanced UNIX system, it appears to the user much like any other UNIX-based system. To execute any program (data parallel or message passing), simply log in to (or `rsh` on to) a partition manager for which you have appropriate privileges and proceed in the familiar way:

```
% a.out
```

You can also use Network Queuing System (NQS) batch processing, which is described in the document *NQS for the CM-5,* Version 2.0.

To obtain information about other processes currently running on a partition, use the command `cmps`. This command, closely modeled on the UNIX `ps` command, reports user names, process IDs, time allocated to the processes, and memory usage on the nodes and vector units. For comparable information on the partition manager itself, use the `ps` command.

2.3 CM-5 I/O Facilities

CM-5 software supports three file systems:

- SFS, the scalable file system, provides access to the CM-5's high-performance scalable disk array (SDA).
- CMFS, the Connection Machine file system, provides access to all I/O devices that are shared by the CM-5, CM-2, and CM-200. These devices (sometimes called the CMIO-bus devices) include the DataVault mass storage system, the CM-HIPPI, and VME-based devices.
- The standard NFS-mounted UNIX file system.

The CM Fortran Utility Library, C* I/O procedures, and CMMD I/O routines allow applications to access disk files in any of these file systems. These procedures allow applications to open, close, truncate, and seek within files. They also allow applications to read and write data in parallel streams between the processing nodes and the SDA or DataVault.

In addition, UNIX files can often be accessed by standard UNIX commands, while special CMFS library procedures enable you to manipulate files stored on devices accessible via VME or CM-HIPPI.

This flexibility in file access, together with the fact that the CM-5 writes files in canonical UNIX order, allows the CM-5 to share data with other machines in a heterogeneous environment.

2.4 Other User Facilities

Other OS facilities available to the user are summarized in connection with CM Fortran programming (Section 4) and CMMD programming (Section 8). See also the description of the Prism programming environment (Section 3).

3 The Prism Programming Environment

Prism is an integrated, graphical environment within which users can develop, execute, debug, and analyze the performance of programs written for the Connection Machine system. It provides an easy-to-use, flexible, and comprehensive set of tools for performing all aspects of CM programming.

Prism operates on terminals or workstations running the X Window System. It is a multiwindow environment, where you choose options from menus with a mouse. You can also call up an editor and a UNIX shell within Prism.

Thus, you can start from scratch within Prism to write and compile your program. Alternatively, you can load an executable program into Prism and proceed from there to debug and analyze it.

Once you have an executable program within Prism, you can (among other things):

- Execute the program (or single-step through it), with the ability to interrupt execution at any time.
- Debug the program by performing **dbx**-like operations such as setting breakpoints, printing variable values, and moving through the call stack.
- Analyze the program's performance at a granularity as fine as procedures or lines of source code.
- Visualize data by specifying variables or expressions to be displayed in a variety of textual and graphical formats.

Prism also provides an interface to on-line documentation for the CM system. You can call up a man page for a CM command or library routine, or view the portions of the CM documentation set that pertain to your current task.

3.1 Documentation Provided

- *Prism User's Guide*
- *Prism Reference Manual*
- *Prism Quick Reference Guide*
- A comprehensive help system, including an on-line tutorial

4 Programming in CM Fortran

Fortran for the Connection Machine system is standard Fortran 77, supplemented with the array-processing extensions of the ANSI and ISO standard Fortran 90.

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

Version 2.1 of CM Fortran (in Beta test as of February 1993) also supports Fortran 90 pointer arrays and 64-bit integers.

Newly written Fortran programs can use these array extensions to express efficient data parallel algorithms for the CM. These programs will also run on any other system, serial or parallel, that implements Fortran 90. CM Fortran also offers several extensions byeond Fortran 90, such as the **FORALL** statement and some additional intrinsic functions. These features are well known in the Fortran community and are particularly useful in data parallel programming.

Many existing Fortran 77 programs can be converted into CM Fortran with the assistance of the CMAX convertor, discussed in Section 9.

4.1 Programming Models

Version 2.1 of CM Fortran on the CM-5 supports both data parallel and message-passing programming styles. (Earlier versions support only the data parallel programming style.)

- When used alone, CM Fortran operates in a "global" manner, as it does on the CM-2. That is, it handles scalar data on the partition manager, but lays out parallel arrays across all the processing nodes of the CM-5 partition on which it executes. (CM-2 users should note that the compiler's view of the CM-5 hardware is identical to the slicewise execution model on the CM-2.) Parallel data is laid out across the processing elements, with each element executing elemental code on its local data, independently of the others.

The control processor (partition manager) executes scalar code and calls run-time functions for interprocessor communications.

Within this basic pattern, two execution models are supported:

- The "VU model" makes use of the optional vector units on the CM-5. It lays out and processes parallel arrays on all the vector units within the partition.
- The "nodes model" ignores the vector units. It lays out parallel arrays in SPARC memory across the processing nodes, and uses the SPARC microprocessor within each node to operate on them.

The compiler switches **-sparc** and **-vu** select the desired execution model.

- When used within a CMMD message-passing program, CM Fortran operates in a "local" manner. Independent copies of the CM Fortran program run on every node. Scalar data is handled by the microprocessor on the node; parallel data is handled by either the microprocessor or the VUs, depending on the available hardware and on the compiler switch chosen. Communication among processors is handled by the application via CMMD message-passing routines.

4.2 Intrinsic Functions

Fortran 90 defines a rich set of intrinsic functions that take an array object as argument and use parallel computation to construct a new array (or scalar). Intrinsics include reduction intrinsics (such as **SUM** and **MAXVAL**) and parallel prefix (or scan) operations; array construction functions such as **TRANSPOSE**, **RESHAPE**, **PACK**, **UNPACK**, and **SPREAD**; and array multiplication functions (**DOTPRODUCT** and **MATMUL**). In addition, CM Fortran offers such instrinsic functions as **DIAGONAL**, **REPLICATE**, **RANK**, **PROJECT**; bit intrinsics such as **FIRSTLOC**, **LASTLOC**, **LEADZ**, **POPCNT**, and **POPPAR**; and a set of intrinsics (new with Version 2.1) that provide a form of "equivalence" (or storage association) on array subgrids.

4.3 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 an interface from CM Fortran to lower-level software such as run-time functions. It also provides the CM Fortran interface to I/O. Utility Library I/O calls can access any CM file system.

4.4 Porting CM Fortran Programs between CM Platforms

Programs written in the CM Fortran language run on both the CM-5 and the CM-2/200. You need not make any changes in the use of language features to port a program from one platform to another. Some other system features, however, are platform-dependent. For example, send addresses on the CM-5 are 8-byte integers. Arrays that contain send addresses, therefore, should be declared (on any CM) as `DOUBLE PRECISION` or `REAL*8`. Send address arrays declared in this way are portable across all CM platforms. Also, some compiler switches apply to only one platform. Optimizations are the same for the CM-5 and the slicewise model on the CM-2/200; Paris optimizations do not apply to the CM-5.

Assemblers also differ: the Paris instruction set is not supported on the CM-5, so calls to Paris must be removed from CM-2 programs before they are ported to the CM-5. Such calls are typically replaced by calls to the Utility Library. Other library calls may also be non-portable:

- CMMD and CMX11 are supported only on the CM-5.
- CM-2 visualization libraries are supported only on the CM-2/200.
- The CM Fortran Utility Library, CMSSL, and CM/AVS are supported on all CM systems.

For futher information on porting programs from the CM-2 to the CM-5, see the *CM Fortran User's Guide*.

4.5 Development and Monitoring Facilities

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

Timing of CM Fortran programs is accomplished using the CMOST timing functions, such as `CM_timer_start`, `CM_timer_print`, and so on. These functions behave as they do on the CM-2, controlling up to 64 separate timers. (They can, however, accumulate significantly more time than the CM-2 timers.)

Run-time safety is enabled by the `cmf` compiler switch `-safety` or (for a subset of safety checks) `-argument_checking`.

4.6 Documentation Provided

- *Getting Started in CM Fortran*
- *CM Fortran Programming Guide*
- *CM Fortran Reference Manual*
- *CM Fortran Utility Library Reference Manual*
- *CM Fortran User's Guide*
- *CM Fortran Optimization Notes: Slicewise Model*
- *CM Fortran Array Operations Quick Reference Guide*
- On-line manual pages for all CM Fortran intrinsic functions and Utility Library procedures.

5 Programming in 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 `%`).

On the CM-5, C* also implements the `*=` and `/=` parallel-to-scalar reduction operators.

As a binary reduction operator, `*=` multiplies the values of the active elements of the parallel RHS by the value of the scalar LHS and assigns it to the LHS. As a unary operator, it returns the product of the active elements of the parallel variable.

As a binary reduction operator, `/=` divides the value of the scalar LHS by the product of the parallel RHS and assigns the result to the scalar LHS. When it is used as a unary operator, it returns the reciprocal of the product of all active positions in the parallel variable.

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.

The C* language on the CM-5 is identical to C* on the CM-2. There are, however, some implementation differences of which programmers should be aware. These

differences are detailed in the *C* Release Notes* for Versions 7.0 and 7.1. The most important ones are as follows:

- The CM-200 C* restrictions on shape extents are not present in CM-5 C*. The only restriction is that the size of each dimension must be greater than 0. (Note that on a CM-5 with vector units, the size of the physical shape is the number of vector units in the partition, not the number of nodes.)
- On the CM-5, parallel `bools` occupy 1 byte of storage, not 1 bit, as on the CM-2 and CM-200. The semantics of using bools remain the same; you need not change an existing program to deal with the new size. Memory usage will go up on the CM-5, however.
- C* on the CM-5 supports parallel enums.
- Because the CM-5 C* compiler is generally compliant with the ANSI standard, it will reject some programs that previously compiled without error.
- CM-5 C* programs cannot call Paris routines.

In addition, the `cs` command has several new options for use on the CM-5, but does not accept certain CM-200 options. Again, the release notes provide details.

- The `-vu` option specifies compilation for a CM-5 with vector units. The `-sparc` compiler option specifies compilation for a CM-5 without vector units.

Version 7.1 of C* (in Beta test as of March 1993) supports node-level (message-passing) programming, in the same manner as does CM Fortran Version 2.1.

5.1 Documentation Provided

- *C* Programming Guide*
- *C* User's Guide*
- Release notes for the version of C* that you are using

6 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. 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. The user interface for these 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.

Versions 3.0 and 3.1 Beta of CMSSL (available as of February 1993) concentrate on six critical areas of programming: numerical linear algebra, Fourier Transforms, ordinary differential equations, linear programming, random number generation, and statistical analysis.

6.1 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, 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 gradient, bi-conjugate gradient with stabilization, quasi-minimal residual, and restarted generalized minimal residual methods.

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

6.2 FFTs

CMSSL offers routines for the computation of Fourier Transforms by Cooley-Tukey type algorithms on one or more axes of arrays with an arbitrary number of axes. Currently, a complex-to-complex FFT is provided. Real-to-complex and complex-to-real FFTs are planned for a future release.

6.3 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.

6.4 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.

6.5 Random Number Generation

CMSSL provides two random number generators. Both use a lagged-Fibonacci algorithm to produce a uniform distribution of random values. Both may be reinitialized for reproducible results.

6.6 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 facilitates breaking data from particularly large data sets into subranges, perhaps as a preliminary step before doing more detailed analysis of interesting areas.

6.7 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 broadcast, 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 (or trace) just once, and then use the trace many times in subsequent data motion and combining operations.

7 Visualization Programming

7.1 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.

7.2 An Integrated Environment

By using the distributed graphics strategy described above, together with an underlying protocol such as X11 or an existing GUI such as AVS, 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 framebuffers, workstations, or X window terminals.

7.3 Visualization Programming with CMX11

The CMX11 visualization library is designed for distributed graphics programming in a heterogeneous computing environment. This library, callable from CM Fortran and C*, allows you to display data from CM-5 memory on an X windows server screen anywhere on your network.

The basic capabilities of drawing and display require only a few subroutine calls and no direct X programming. Parallel extensions of X routines are provided for those who wish to do more elaborate graphics programming.

7.3.1 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 environmental 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.

7.3.2 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.

- Before drawing to a display, use CMX11 functions to get the identifiers for the underlying X display, drawable, and graphics context. These values are passed as arguments to the drawing routines.

- Call one or more CMX11 drawing routines, such as `CMXDrawPoint`. The arguments are parallel arrays, in this case arrays of point coordinates. With this one call, the entire set of points is drawn in the window's foreground color.

  Another version of the drawing routines, for example, `CMXDrawCPoint`, accepts a parallel array of color values in addition to point coordinates. These routines enable you to specify a color for each point.

7.3.3 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.

7.3.4 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 for Version 1.5*
- On-line man pages for all CMX11 routines

For more elaborate graphics programming, users who are unfamiliar with X may wish to consult the following publications in addition (these are provided along with the CMX11 library):

- *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)

7.4 The CM/AVS Visualization Environment

CM/AVS is the first of the GUIs available on the CM-5. Other GUIs are expected to be adapted to the CM-5 in the future.

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

7.4.1 Documentation Provided

- *CM/AVS User's Guide*
- *CM/AVS Release Notes for Version 1.0*
- On-line man pages viewable through the AVS man page viewer.

In addition, users should have the AVS document set.

8 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, C++, and Fortran 77; programs are compiled with the appropriate Sun compiler. At Version 3.0, CMMD is also callable from CM Fortran, with the program compiled by the `cmf` compiler, and from C* with the program compiled by the `cs` compiler. Programs or program modules written in CM Fortran or C* utilize the vector units on the CM-5. Other programs or modules use only the microprocessor on each node.

8.1 Programming Models

CMMD Versions 2.0 and 3.0 support two programming models:

- The host/node programming model involves two simultaneously running programs. 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 control processor 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 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.

8.2 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.

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 pre-defined array into the channel; the receiving node reads the channel, then resets it for another use. No synchronization is needed.

8.3 Remote Memory Access and Active Messages

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 remote memory access and the Active Message facility.

- With remote memory access, one node reads or writes a portion of a second node's memory, as if the two nodes shared a common memory.

- 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.

8.4 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.

8.5 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 the **pndbx** debugger (used either independently or from within Prism). The *CMMD User's Guide* provides hints for using program development and monitoring facilities.

8.6 Documentation Provided

- *CMMD Reference Manual*
- *CMMD User's Guide*
- On-line man pages for all CMMD routines

9 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 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 provides a convenient interface to the user. The Prism development environment provides facilities for examining CMAX output and comparing it line-by-line with the input program. 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.

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.

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 convention) 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.

10 Where to Find Out More

Because so much of the data parallel software is shared across machines, the CM-5 documentation set is made up partly of documents specifically for the CM-5 and partly of documents shared with the CM-2 and CM-200.

CM-5 Overviews

CM-5 Technical Summary
CM-5 Software Summary, Version 7.2
CM-5 System Release Notes, Version 7.2

CMOST Operating System and I/O

NQS for the CM-5, Version 2.0
CM-5 I/O System Programming Guide, Version 7.2
Using the CM-HIPPI on the CM-5, Version 7.1
CM-5 System Administrator's Guide, Version 7.1
CM-HIPPI User's Guide for the CM-2, Version 6.1 Beta 2
CM I/O System Programming Guide, Version 6.1

Prism Development Environment

Prism User's Guide, Version 1.2
Prism Reference Manual, Version 1.2
Prism Release Notes, Version 1.2
Prism Quick Reference Guide, Version 1.2
An on-line tutorial as part of the Prism help system

CM Fortran Data Parallel Language

CM Fortran User's Guide for the CM-5, Version 1.1.3 (CMOST V 7.1), or
CM Fortran User's Guide, Version 2.0 Beta
Getting Started in CM Fortran
CM Fortran Reference Manual, Version 1.1 or Version 2.0 Beta
CM Fortran Utilities Reference Manual, Version 2.0 Beta
CM Fortran Programming Guide, Version 2.0 Beta

CM Fortran Array Operations Quick Reference Guide, Version 2.0 Beta
CM Fortran Release Notes, Version 2.0 Beta

C*

C User's Guide*, Version 6.0.2
C Programming Guide*, Version 6.0.2
C Release Notes*, Version 7.1. Beta

CMSSL

CMSSL for CM Fortran: CM-5 Edition, Version 3.1 Beta

CM-5 Visualization Products

CM/AVS User's Guide, Version 1.0
CM/AVS Release Notes, Version 1.0
CMX11 Reference Manual, Version 1.2
CMX11 Release Notes, Version 1.2
(NOTE: The CM-2 graphics documentation does not pertain to the CM-5.)

CMMD Message-Passing Library

CMMD Reference Manual, Version 2.0 or Version 3.0 Beta
CMMD User's Guide, Version 2.0 or Version 3.0 Beta
CMMD Release Notes, Version 2.0 or Version 3.0 Beta

CMAX Converter

Using the CMAX Converter, Version 0.8 Beta

*Lisp Data Parallel Interpreter

*Porting to CM-5 *Lisp,* Version 7.1
*Getting Started in *Lisp*
**Lisp Dictionary,* Version 6.1