Chapter 2

Execution Models

2.1 CM-5 Architecture

A CM-5 system consists of a large number of processing nodes, a small number of partition managers (PMs) and some number of I/O control processors (IOCPs) and I/O devices. These components are all linked together by two internal communications networks, the Data Network and the Control Network. (See Figure 1.)

The Data Network is used for fast, high-bandwidth communication of data between the processing nodes, PMs, and I/O devices. The Control Network provides a number of global control operations, such as broadcasts, scan/reduction operations, and node synchronization.

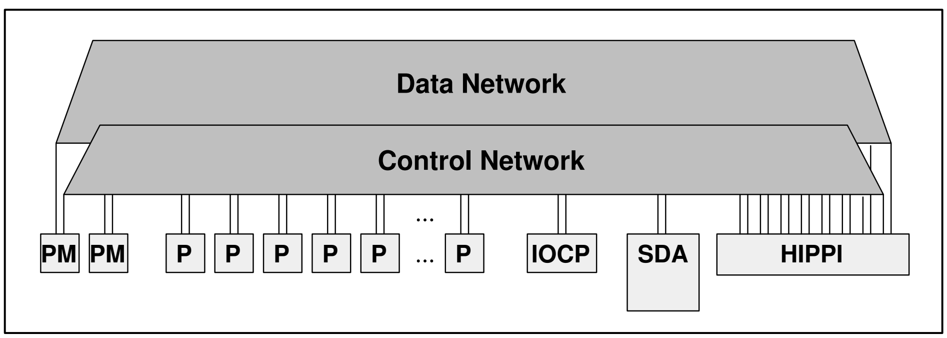

Figure 1. Components of a typical CM-5 system.

Each processing node contains a RISC microprocessor and a Network Interface (NI) chip that connects it to the networks. Each processing node may also contain four vector processor units (VUs), which are located between the RISC microprocessor and the node's memory. The VUs, when they are present, provide highly efficient memory-based arithmetic calculations.

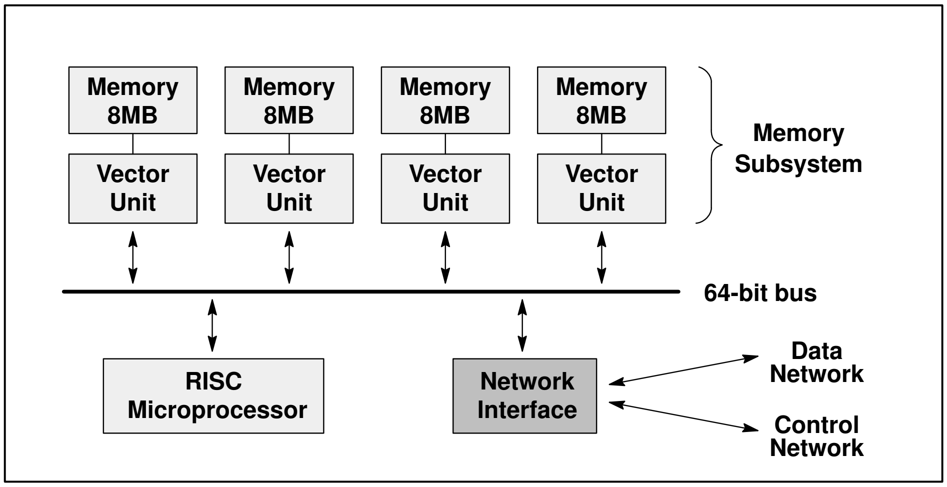

Figure 2. Components of a CM processing node.

The CM-5 processing nodes are grouped into one or more partitions, each controlled by a partition manager. Program execution always begins and ends on the partition manager. While the program is running, the PM may, depending on the programming model in use, either direct the actions of the processing nodes or serve as a silent I/O server for the nodes, providing access to the other devices attached to the CM-5 network.

The CM-5 system uses a parallel timesharing operating system known as CMOST, which allows each partition of nodes to execute as a separate parallel timesharing system. Each partition has its own set of processes to execute, and can be considered a separate parallel processing system, except when it needs to communicate with other partitions and with I/O devices.

2.1.1 Logical Addresses

Each node within a partition has a *logical address* relative to that partition, from 0 to (partition-size –1). The host (PM) also has a logical address, outside the range of node addresses.

The nodes (and the host) use these logical addresses when sending messages. They use three functions to gain the requisite information: **`CMMD_self_address`**, **`CMMD_partition_size`**, and **`CMMD_host_address`**. See Chapter 4 for information on these functions.

2.2 The CM-5 Execution Model

An application program executing on a CM-5 is actually two separate programs. One program runs on the partition manager (the "host" in CMMD terminology); the other runs on all nodes. Frequently, this duality is transparent to programmers: a programmer simply writes, for example, a global CM Fortran application, or a "hostless" message-passing program, and the CM software does the work required to create the appropriate host and node executables. Some cases, however, such as host/node programming or programming in DPEAC, require specific actions from the user to define and coordinate host and node code.

In a CM-5 message-passing program:

- The program executes in multiple copies, one on each processing node. Although all nodes execute the same program, each node operates independently of the others.

- The nodes explicitly handle all load balancing, distribution of tasks, etc. Each node determines for itself how its data will be laid out.

- Communication between nodes is in the form of "messages" sent from one node to another, or from one node to many nodes. It is with these messaging features that the CMMD library of functions is mainly concerned.

- The partition manager typically does nothing more than start the program running on each of the nodes, and thereafter acts as an I/O server for the nodes. This is the standard, "hostless" form of message-passing.

- Alternatively, the partition manager may execute a separate "host" program that directs the actions of the nodes, distributes data to them, and collects results. This is the "host/node" form of message passing.

- I/O may be handled by each node independently, with each node opening and closing its own files. CMMD also provides functions for opening "global" files that can be accessed either by any node independently or by all nodes simultaneously.

2.2.1 Hostless and Host/Node Programs

To create a hostless program, a user writes a single program that will execute on all nodes. CMMD supplies the server program that executes on the host.

To create a host/node program, a user writes two separate programs, one for the "host" (the partition manager), and one for the nodes. This method of CMMD programming is described in Chapter 16 and in the *CMMD User's Guide*.

2.2.2 Memory Maps

Both host and node programs look like standard UNIX executables. Each has its text region, heap, multiple stacks, and so on. The extent of each of these regions is maintained uniformly across all nodes.

The address of any function is therefore identical on all nodes (but *not* on the host.). This allows one node to trigger execution of a function on a second node, as in the case of active messages and their handler functions.

2.3 Languages

CMMD programs can be written in standard Fortran 77, C, or C++. Programs written in these languages execute exclusively on the microprocessor in each node, ignoring the vector units. They exchange data exclusively in the form of serial arrays (that is, blocks of microprocessor memory). (See Figure 3.)

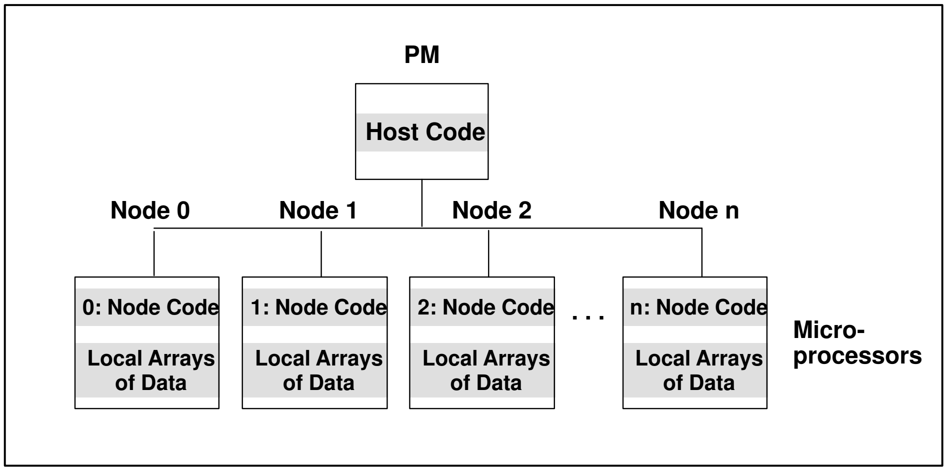

Figure 3. Execution model without VUs.

CMMD programs can also be written in the CM's data parallel programming languages*, CM Fortran and C*. Programs written in parallel languages use both the microprocessor *and* the vector units; they store serial arrays in microprocessor memory and parallel arrays in VU memory. (See Figure 4.) Similarly, programs written in C, C++, or Fortran 77 can call modules written in DPEAC assembly language (or in C/DPEAC), in order to access VU memory.

A parallel array in a "node-level" CMMD program is stored on a single node, with its values spread across the memory of the node's four vector units. It can thus take advantage of the VUs for high-speed floating-point computations. These parallel arrays can be passed as arguments to CMMD functions in much the same way as serial arrays.

---

* In Connection Machine manuals, the term *data parallel* implies a relatively synchronous parallel programming model featuring a single conceptual thread of control and a global approach to data layout and I/O, in which arrays of data are laid out across all processors, and all elements are computed upon simultaneously, by their respective processors. Although the conceptual model itself is completely synchronous, in practice processors may execute asynchronously any operations that are independent of other processors, such as conditional operations on locally stored data. Connection Machine compilers take advantage of this freedom to exploit the MIMD hardware capabilities of the CM-5.

A *data parallel language* contains syntax that facilitates operating on all the elements of an array at once: for example, arrays are treated as first-class objects; array operations such as circular shifts and matrix multiplication may become intrinsic functions; and such constructs as the Fortran **FORALL** specify independent processing of assignment statements at each position of an array. In support of such languages, the Connection Machine data parallel compilers, run-time system, and file systems deal automatically with global layout and communications operations, while its DPEAC assembler provides the programmer with direct control of the vector units.

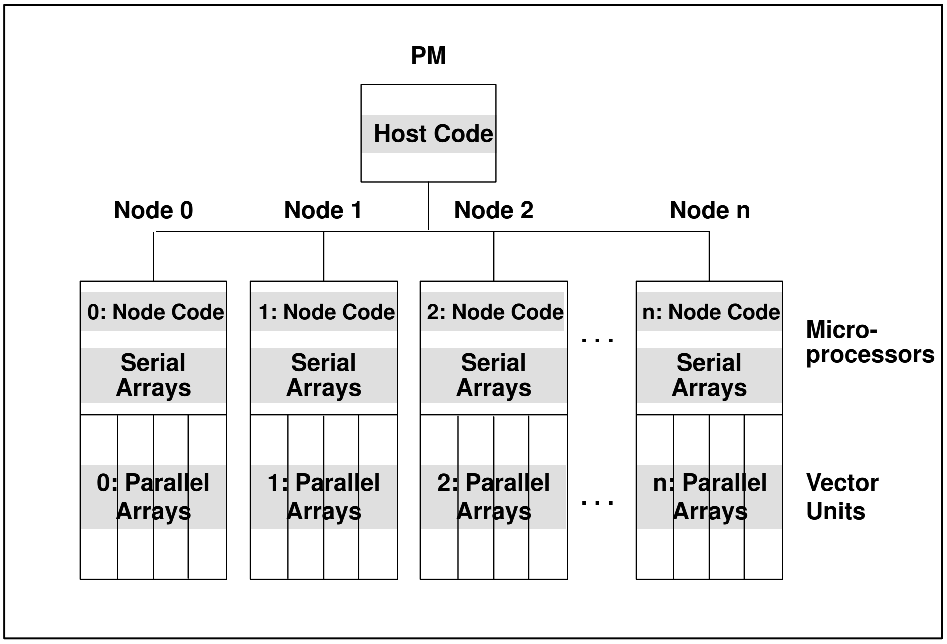

Figure 4. Execution model with VUs.

2.3.1 CM Fortran

CM Fortran users should note the distinction between a "global CMF" program (in the traditional style) and "CMF on a node" with CMMD.

In a global CMF program,

- serial arrays (FE arrays) are stored on the PM
- parallel arrays (CM arrays) are global, laid out across all VUs in the partition.

In a nodal CMF program,

- serial arrays (FE arrays) are stored on each node
- parallel arrays (CM arrays) are local and are laid out across the node's four vector units.

At Version 3.0, certain restrictions apply to CM Fortran programs:

- CM Fortran programs are most easily written in the hostless model. If the host/node model is used,
  - the host program cannot be written in CM Fortran
  - If CM Fortran is used for the main node program in the host/node model, special linking techniques (explained in the *CMMD User's Guide*) may be required.
- I/O is limited. See Chapter 12 for details.

For the Curious: A "global/local" programming model for CM Fortran, combining the two approaches, is planned for a future release, but is not yet available.

2.3.2 C*

Like "CMF on a node," "C* on a node" distributes each shaped array across the four VUs of a single node; the arrays are local, not global, in nature.

Unlike CM Fortran, C* requires a slightly different syntax for sending or receiving a parallel array than for sending or receiving a serial array. This syntax is explained in the discussions of the relevant functions later in this manual. C* uses the variables' declaration to distinguish serial from parallel arrays.

Programs written in C* should use the hostless model. The main node programs must be written in C*.

2.3.3 DPEAC

C and Fortran code written for CM-5 systems can include assembly-language routines. The assembly language for CM-5 systems with VUs is DPEAC. DPEAC instructions allow a node's VUs to operate in parallel on separate data. All four VUs (or some subset) perform the same operation at the same address, each in their individual memory bank.

An application program module coded in DPEAC uses OS function calls to allocate parallel memory, then uses DPEAC instructions to manipulate data within that memory. The address returned by the allocation instruction may be passed as an array address to any CMMD send or receive function. The syntax is the same as if a block serial function were being transmitted; the CMMD system software determines from the array's address whether the array is indeed a block serial array or whether it is an array in VU memory.

DPEAC instructions, and their use, are documented in the *DPEAC Reference Manual*. The use of DPEAC from CMMD is discussed in the *CMMD User's Guide*.