## Chapter 3

# Execution Models

CM Fortran's *execution model* refers to the way a program makes use of the hardware. CM systems support three basic execution models, plus some variations:

- *global* model, where a single program operates on arrays of data spread across all the parallel processors
- *nodal* model, where multiple copies of a program operate independently on subsections of the data and communicate in message-passing style
- *global/local* model, a combination of data parallel and message passing for maximum flexibility in programming
- *simulator*, where a data parallel program executes on a single processor (for purposes of testing and debugging)

The compiler always generates both serial and parallel code for its built-in "two-machine" model of the hardware. A user-supplied switch determines which system components are to serve as "the front end" and "the CM." These switches control the assembly-level language generated for parallel code blocks and determine the libraries used for linking.

**NOTE**

The execution models are not object-code-compatible. If you compile and link separately, be sure to specify the target model on both command lines.


*Copyright © 1989–1994 Thinking Machines Corporation*

## 3.1 Global Data Parallel Models

The global models describe a program executed by a single control processor and a set of system components that serve as parallel processors. The parallel processors may be either the nodes or the vector units.

The global models are the most elegant and convenient for programming, since the compiler generates code that transparently handles parallel memory allocation, array layout, and interprocessor communication. You can take some control over allocation and layout, by using dynamically allocated arrays and by supplying **LAYOUT** directives to the compiler. You cannot, however, control communication except by using different language constructions. Synchronization is always automatic in the global models: the processors synchronize after every block of parallel computation and every communication operation.


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

### 3.1.1 CM-5 Global SPARC Nodes Model

In this model, the partition manager serves as the control processor, and the node microprocessors are the parallel processors. If the system has vector units, they serve merely as memory controllers and do not participate in the processing of parallel data. Code for parallel computations is generated in SPARC assembler.

To compile and link for the SPARC nodes model:

```
% cmf -cm5 -sparc myfile.fcm
```

Arrays are allocated in the expected data parallel fashion, as shown in Figure 7: front-end arrays on the partition manager and CM arrays across the nodes. The figure shows a 16-element vector **A** laid out on four nodes. A 4-element vector would be laid out one element per node.

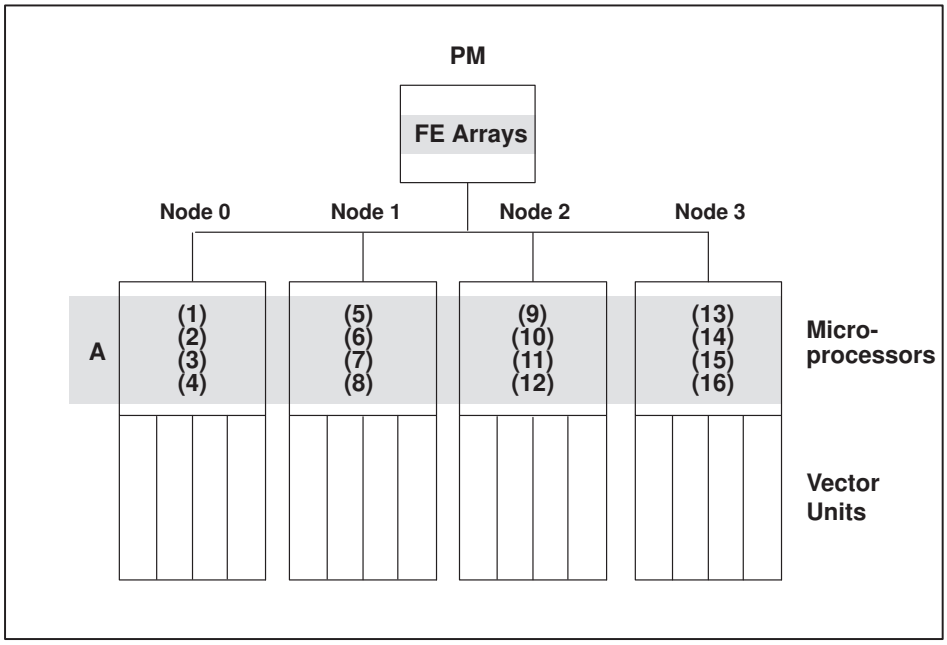


**Figure 7. SPARC nodes model.**


Copyright © 1989–1994 Thinking Machines Corporation

### 3.1.2 CM-5 Global Vector-Units Model

In this model, the partition manager serves as the control processor, and the vector units are the parallel processors. The SPARC microprocessors on the nodes are invisible to the program, although they assist the vector units with OS services and communication. Code for parallel computations is generated in DPEAC assembler.

To compile and link for the vector-units model:

```
% cmf -cm5 -vu [-nopadding] myfile.fcm
```

Arrays are allocated as shown in Figure 8: front-end arrays on the partition manager and CM arrays across the vector units. The figure shows a 16-element vector **A** laid out across 16 vector units (4 nodes).

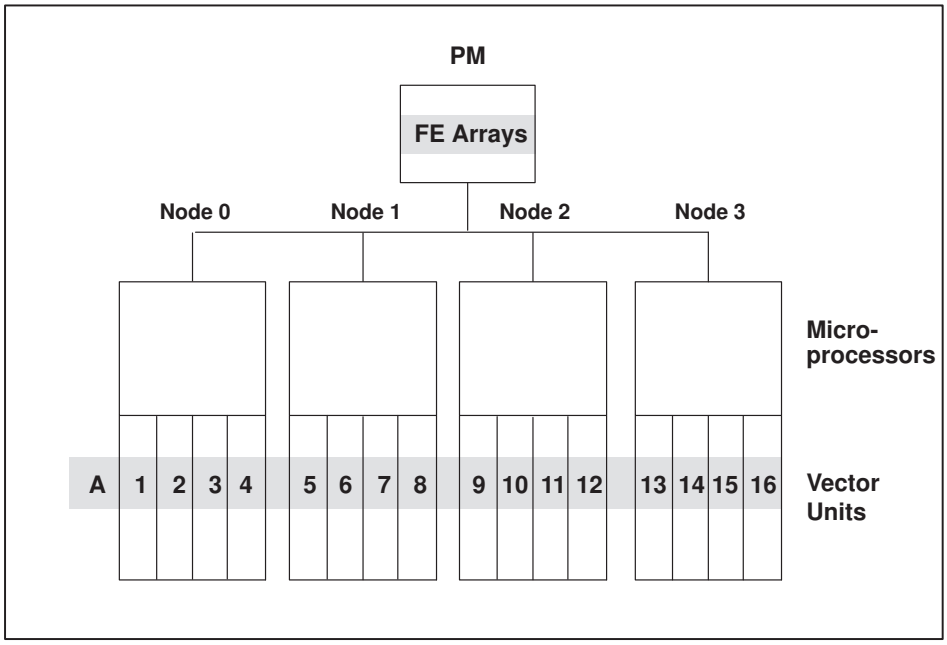


**Figure 8. Vector-units model without vector-length padding.**

The significance of the **-nopadding** switch shown in the command line is that it permits any size array or dimension to be spread evenly across the vector units, as in Figure 8. Without this switch, the system allocates parallel memory in quanta of eight elements per vector unit, the length of the vector


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

register. If an array's size is not a multiple of 8 times partition size, the unused memory at the high end of one or more axes contains garbage data or *padding*, as shown in Figure 9.

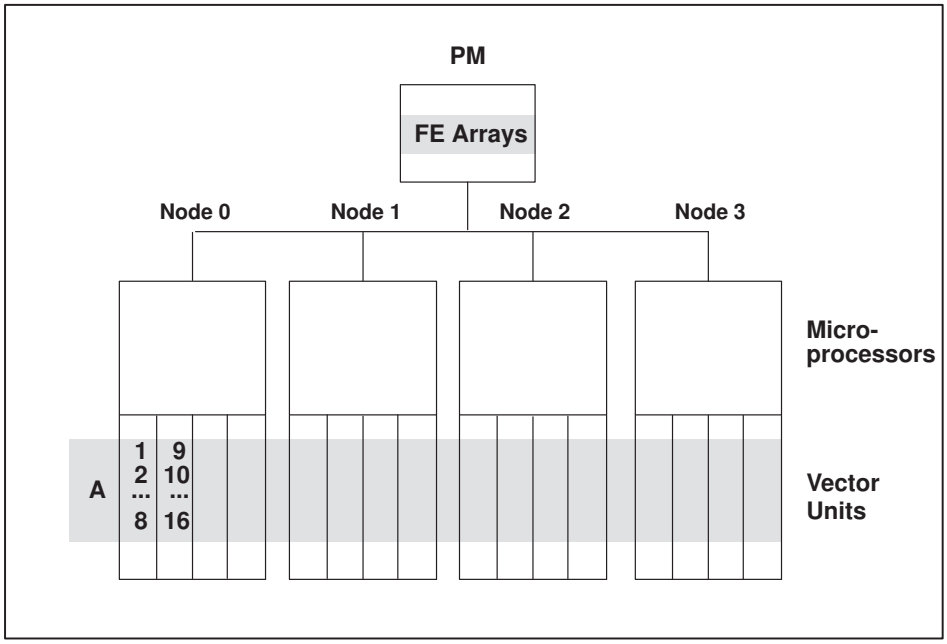


**Figure 9. Vector-units model with vector-length padding.**

Padding is not a problem for program correctness, since the system masks it out of any communication or store operation. It can, however, waste system resources and make for undesirable array layouts, as it has with the 16-element vector. Since the system uses block layout, the first two vector units contain all the data, and the others contain garbage. On 16 vector units, array size needs to be a multiple of 128 elements (8 x num-procs) to be stored without padding. See Chapter 12 for more information on CM array layouts.


*Copyright © 1989–1994 Thinking Machines Corporation*

## 3.2 Node-Level Message-Passing Models

The nodal models describe the independent execution of multiple copies of a CM Fortran program on the nodes of a CM-5. For each copy, the microprocessor in the node serves as the CM Fortran "front end," and the four vector units serve as its parallel processors. The programs use the CM message-passing library CMMD.

The nodal models are appropriate for a more do-it-yourself style of programming than the global models. The programmer has the convenience of using CM Fortran features to allocate and compute on arrays, but uses CMMD library calls to cause the nodal programs to communicate and synchronize with one another as needed. CMMD also provides routines for parallel I/O for all the nodes cooperatively, since CM Fortran in the nodal models supports only serial I/O from individual nodes acting independently.

### 3.2.1 CM-5 Nodal Model (Hostless)

In this model, the node microprocessor serves as the control processor and its four vector units are the parallel processors of a CM Fortran program. Code for parallel computations is generated in DPEAC assembler. Communication between programs is explicit via CMMD calls.

To compile and link for nodal CM Fortran (hostless) with CMMD message passing:

```
% cmf -cm5 -vu [-nopadding] -node myfile.fcm
```

```
INCLUDE 'cmmd_fort.h'
```

The division of labor and the allocation of arrays among system components is shown in Figure 10. CMMD provides a server program, running on the partition manager, that starts up copies of the CM Fortran program on each of the nodes. The partition manager is invisible thereafter, serving mainly as an I/O controller. For each program, the node microprocessor stores and processes front-end arrays (called *serial arrays* in CMMD), and the associated four vector units process CM arrays (called *parallel arrays* in CMMD). The programmer decides how to decompose the data among the nodal programs. Each program then allocates its share of the data across its four parallel processors.


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

Recall the multiple-of-8 rule for allocating memory on the vector units (Section 3.1.2). The **`-nopadding`** switch gives you finer control of the allocation of parallel memory.

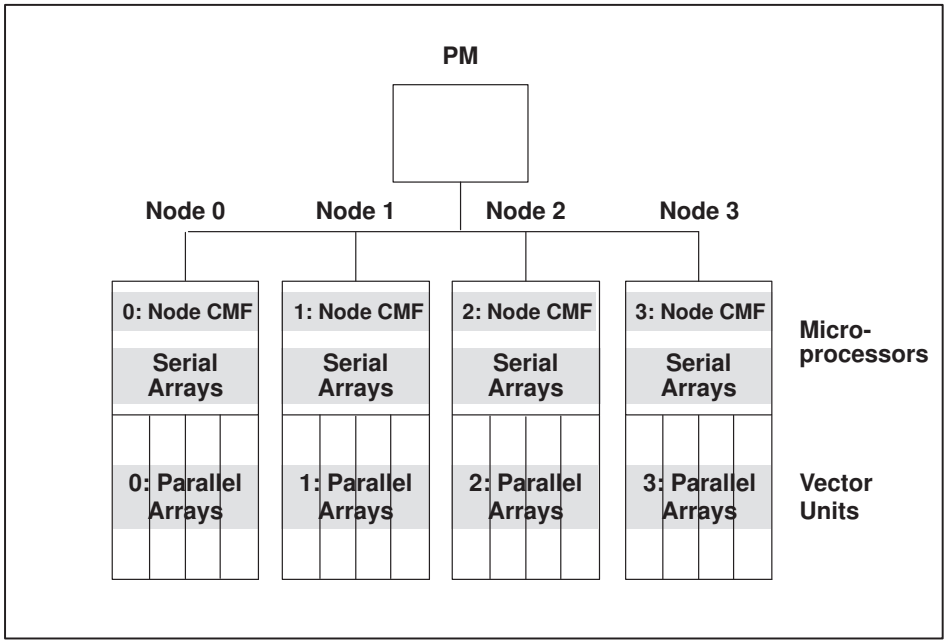


**Figure 10. Nodal CM Fortran model, hostless model.**

### 3.2.2 Fortran 77 on a Node (Hostless)

The nodal model of CM Fortran is clearer if you compare it with the more familiar nodal Fortran 77, supported by the CM-5 and other vendors' systems. In this model, the node microprocessors execute copies of a Fortran 77 program and communicate via a message-passing library (CMMD in the case of the CM system). See Figure 11.


Copyright © 1989–1994 Thinking Machines Corporation

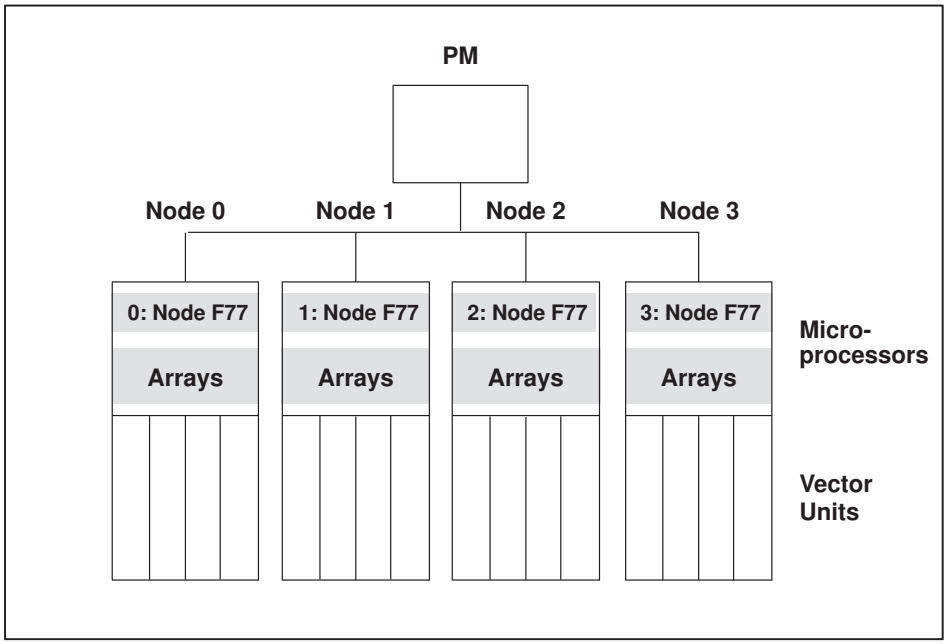


**Figure 11. Nodal Fortran 77, hostless model.**

The nodal program executes entirely on a node microprocessor. It does not get the performance benefits of the vector units, since they can execute only Fortran 90−style array operations encoded in DPEAC assembler.

Nodal programs written in CM Fortran do get the benefit of vector processing. Their array syntax gives the compiler the dependence information it needs to generate parallel computations on array elements, which then execute on the vector units.

Nodal CM Fortran is supported only with the **`-vu`** switch. A nodal **`-sparc`** model of CM Fortran would have the programming convenience of the Fortran 90 array syntax, but there would be no system component to serve as the parallel processors. For this reason, CM Fortran does not support the combination of **`-sparc -node`**.


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

### 3.2.3 CM-5 Nodal Model (Host/Node)

This model is similar to the hostless model of CM Fortran: the node microprocessor serves as the control processor and its four vector units are the parallel processors. Code for parallel computations is generated in DPEAC assembler. Communication between programs is explicit via CMMD calls.

The difference from the hostless model is that the programmer writes the host program, instead of letting CMMD provide it. The host program cannot be in CM Fortran; it must be in a serial language such as Fortran 77 or C.

To compile and link for nodal CM Fortran (host/node) with CMMD message passing, you must specify on the **cmf** command line the files that contain the host code and the appropriate way to link them. Use the **cmf** switches **-host** and **-comphost** for this purpose. For example, assuming a Fortran 77 host program:

```
% cmf -cm5 -vu -node file1.fcm -comphost f77 -host file2.f

INCLUDE 'cmmd_fort.h'
```

In this example, the **cmf** command invokes the **f77** compiler for the **.f** file and, because of the **-comphost** switch, links it with Sun FORTRAN libraries. Each **.f** file must be preceded by **-host**. The execution model switches (**-vu** and **-node**) govern all the files not preceded by **-host**.

The division of labor and the allocation of arrays among system components is shown in Figure 12. It is similar to the hostless model shown in Figure 10, except that the partition manager is running a user-written program and thus may play a more active role at run time.


Copyright © 1989–1994 Thinking Machines Corporation

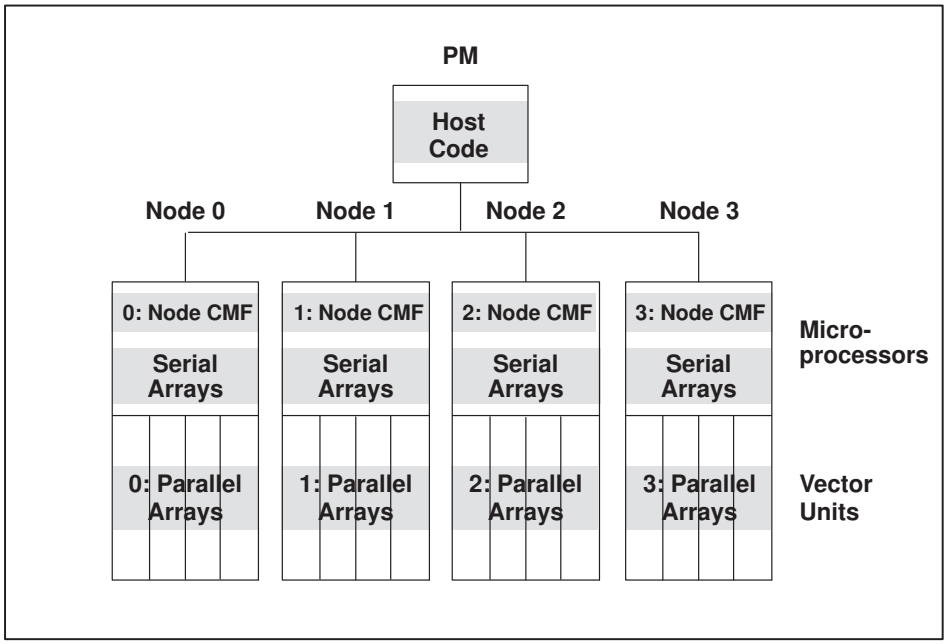


**Figure 12. Nodal CM Fortran, host/node model.**


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

## 3.3 The Global/Local Model

The global/local model describes a CM Fortran program that executes in global data parallel fashion across the entire partition of a CM-5, but which can call local subroutines that run independently on the nodes. A copy of the local routine runs on each microprocessor, using the four associated vector units as its parallel processors. Code for parallel computations is generated in DPEAC assembler, in both the global and local program units. The global program uses transparent compiler-generated communication; the local programs use the CM message-passing library CMMD.

The global/local model combines the convenience of global programming with the fine-tuned control of the message-passing style. The global program allocates memory and distributes CM (parallel) arrays as usual; the local routines then treat their respective *subarrays* of the global array as their own CM arrays, as in the nodal models.

The global program must be in CM Fortran; the local routines can be in either CM Fortran or C (to facilitate direct calls to C/DPEAC). Global/local programming is described more fully in Chapter 13 of this manual.

To compile and link for the global/local model, you must specify on the **cmf** command line the files that contain local code and supply a *prototype* file (described in Chapter 13) that defines the interface between the global and local portions of the program. For example, assuming a CM Fortran subprogram:

```
% cmf -cm5 -vu global.fcm -local local.fcm file.proto

  in local.fcm:   INCLUDE 'cmmd_fort.h'
  in global.fcm:  INCLUDE 'cmgl.h'
```

A separate **-local** switch must precede each file that contains local code.

The division of labor and the allocation of arrays among system components is similar to the host/node model shown above — with one crucial difference: front-end arrays can appear either on the partition manager (in the global program) or on the nodes (for the local routines). The global/local library provides the procedure **CMGL_BROADCAST_SERIAL_ARRAY** to copy a global front-end array to the nodes; the local routines can also create local front-end arrays. CM arrays are distributed across the vector units of the partition. Unlike front-end data, CM array data does not move when the program shifts from the global view to the local view.


Copyright © 1989–1994 Thinking Machines Corporation

The scheme is shown in Figure 13 (assuming no vector-length padding of the array shown). Front-end arrays are allocated on the partition manager in the global program, but explicitly broadcast to the nodes for use as front-end arrays in the local programs. CM arrays are always allocated on the vector units; only the system's view of them changes when the global program calls a local subprogram.

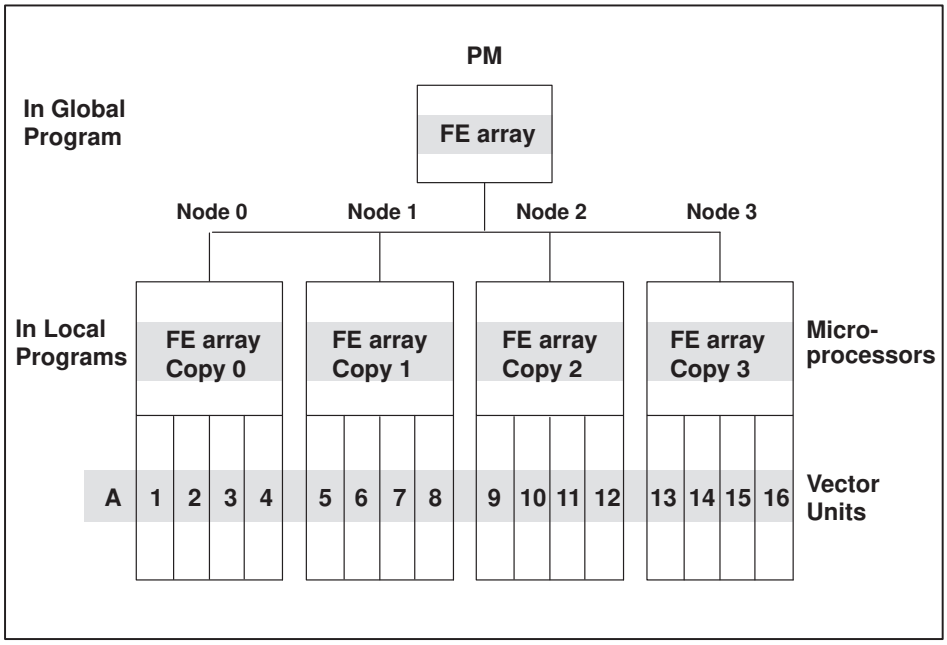


**Figure 13. Global/local model without vector-length padding.**

The global program operates on the entire array, whereas the local routines operate asynchronously on their respective portions of it. For example, consider a 16-element vector **A**, laid out across the vector units of 4 nodes. Figure 14 shows the different effects of calling the circular shift function `CSHIFT` from the global program and from the local routine.


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

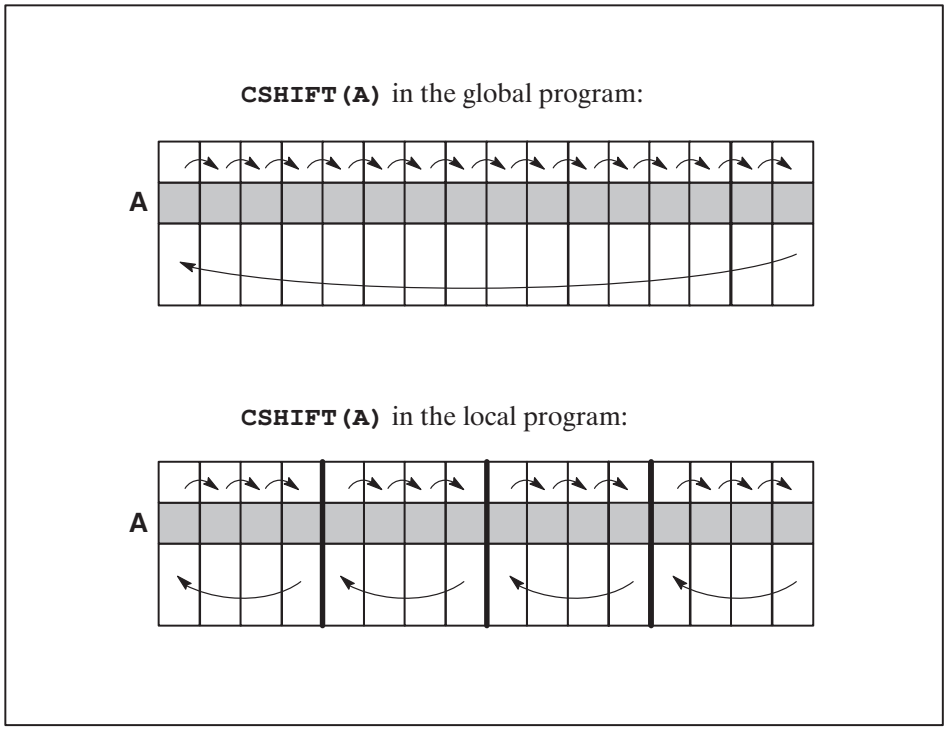


**Figure 14. Computing on arrays and subarrays in global/local model.**

## 3.4 CM Fortran Simulator

In the simulator model, a CM Fortran program executes on a single Sun-4 processor, which may be a CM-5 partition manager or a stand-alone workstation. This model is convenient for developing and debugging programs without tying up parallel processing resources.

To compile and link for the CM Fortran simulator:

```
% cmf -cmsim myfile.fcm
```

Notice that you do *not* specify a hardware platform, like **-cm5**, or the execution model you eventually intend to use for the program, such as **-vu** or **-sparc**. The compiler generates code for its "two-machine" model as usual, but links with a library that causes all operations to be performed sequentially. The simulator does not support message passing.


Copyright © 1989–1994 Thinking Machines Corporation

Since the code is generated for two machines, there is a division of labor of sorts but the Sun-4 processor plays both the serial and parallel roles. The simulator generates two structures in serial memory to serve as the stack and heap memory of parallel processors, as shown in Figure 15. Code for parallel computations is generated in SPARC assembler.

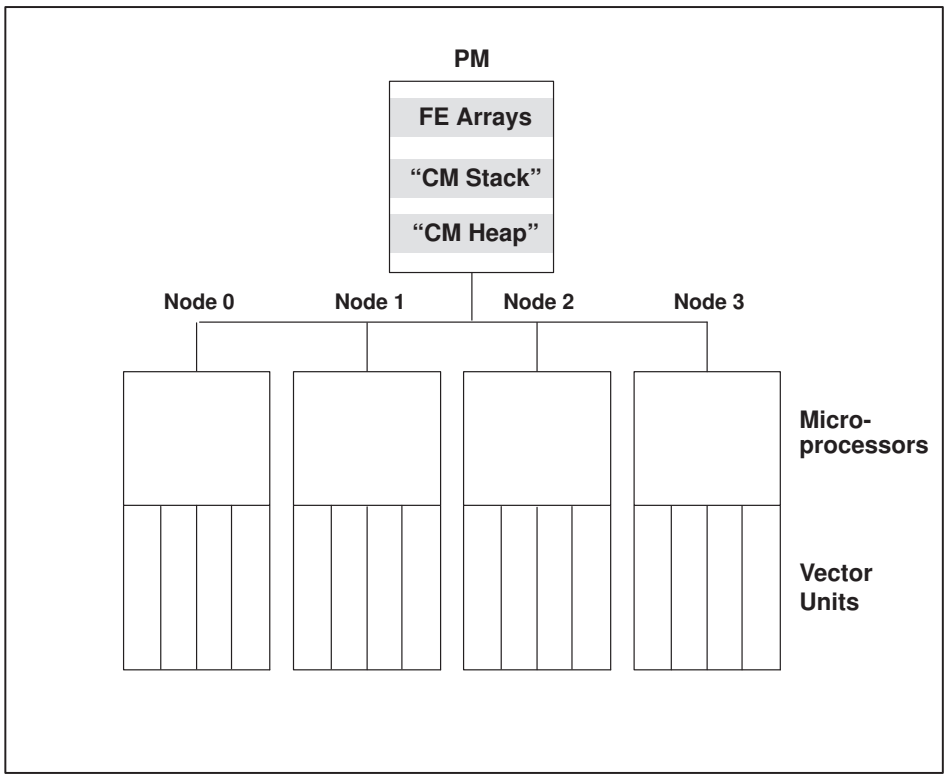


**Figure 15. CM Fortran simulator.**

The CM Fortran simulator is intended only as a convenience for program development. It necessarily runs slower than comparable Fortran 77 code on a Sun-4, since it partitions memory as shown and calls low-level CM runtime functions for simulated "communication." At the same time, since the compiler output is linked for a one-node system, it cannot give the programmer an idea of how fast the same code would run when linked for an *n*-node system.



Copyright © 1989–1994 Thinking Machines Corporation