Addendum to the CM-5 C* User's Guide

Version 7.1, May 1993

1 Physical I/O Routines

Section 2.4.3 of the *CM-5 C* User's Guide* omits a discussion of routines for performing physical I/O; these routines are available in Version 7.1.

The physical I/O routines are **`CMFS_write_file_physical`** and **`CMFS_read_file_physical`**. They have the same interfaces as **`CMFS_write_file_always`** and **`CMFS_read_file_always`**. Namely:

```
int CMFS_write_file_physical(int fd,
                             void:void* source,
                             int bytes_per_position)

int CMFS_read_file_physical(int fd,
                            void:void* destination,
                            int bytes_per_position)
```

These routines are faster than the corresponding standard I/O routines and use less memory. However, note these points about their use:

- They are not portable. If you use the **`CMFS_write_file_physical`** routine to write data to an external storage device, you can recover the data only by using the **`CMFS_read_file_physical`** routine to read the data onto the same set of physical processors, and into a parallel variable with the same memory layout as the parallel variable from which the original write occurred; you must also use the same file system. Essentially, you must use the same partition size, file system, and shape for both writing and reading; if you do a physical write, you must do a physical read.

- These routines return the total number of bytes read or written, rather than the number of bytes read or written per-position.
- The I/O data read or written by these routines is padded. When the file pointer is not located at an alignment boundary appropriate to the I/O device where the file referenced through the **fd** argument resides, the file pointer is adjusted upward to such a boundary before the transfer of the data between the CM and the I/O device begins. Furthermore, garbage data is written to pad the length of the I/O transaction to the same alignment. For DataVault I/O, the alignment padding used is 512 bytes; for SDA I/O; it is 16 bytes (as of CMOST 7.2 Beta 2, it is 16 bytes times the number of disks across which the SDA stripes).

  To anticipate the impact of padding before doing a read or write, use this routine:

```
int CMFS_physical_transfer_length(int fd,
                                  void:void* target,
                                  int bytes)
```

  CMFS_physical_transfer_length takes the arguments of a **CMFS_read_file_physical** or **CMFS_write_file_physical** function call and returns the number of bytes that such a call will transfer. In other words, it returns the same value that a successfully completed call to the read or write function would return.

  Because of the complications of padding, we recommend using a physical read only when the file pointer is in the same position it was in when the physical write was performed.

2 Additional Discussion of Return Values of I/O Read and Write Routines

Section 2.4.3 notes that **CMFS_read_file[_always]** and **CMFS_write_file[_always]** return the number of bytes read or written in each position of the parallel variable. In addition, note that, if the requested amount of data isn't actually read, **CMFS_read_file[_always]** returns 0, because it can't sensibly represent the number of bytes read in each position (which may be fractional).