Chapter 3

CM-5 I/O Performance Issues

The SDA's Scalable File System (SFS) and the DataVault's CM File System (CMFS) are both optimized for large transfers. This chapter describes how to determine whether your application's transfer size is large enough to attain best performance, what to do if the transfer size must be smaller than optimal, and how to time I/O operations.

3.1 Definition of Terms

Disk block — The minimum amount of data the SDA or DataVault hardware can operate on in an I/O operation.

SDA: 512 bytes * (number of SDA data disks)
= 4 Kbytes for an 8-disk SDA
= 8 Kbytes for a 16-disk SDA
= 12 Kbytes for a 24-disk SDA
= 16 Kbytes for a 32-disk SDA

DataVault: 16Kbytes for any DataVault

The **`f_width`** field of the structure returned by **`statfs()`** tells the number of SDA data disks.

Disk block boundary

Occurs every *n* bytes from byte 0, where *n* is the number of bytes in one disk block.

Copyright © 1994 Thinking Machines Corporation

File system block size

The buffer size used when performing serial I/O, i.e., file system ↔ CP (or other serial computer):

SDA: Depends on number of disks:
= 28K for a 7-disk SDA
= 16K for an 8-disk SDA
= 28K for a 14-disk SDA
= 22.5K for a 15-disk SDA
= 16K for a 16-disk SDA
= 22K for a 22-disk SDA
= 23K for a 23-disk SDA
= 12K for a 24-disk SDA
= 23K for a 46-disk SDA
= 17.5K for a 70-disk SDA
= 23.5K for a 94-disk SDA
= 29.5K for a 118-disk SDA
= 35.5K for a 142-disk SDA
= 41.5K for a 166-disk SDA
= 47.5K for a 190-disk SDA
= 53.5K for a 214-disk SDA
= 59.5K for a 238-disk SDA

DataVault: 16 Kbytes

To verify the block size for any CM-5 file system, call **`CMFS_stat`** or **`CMFS_fstat`** for any file in a CMFS file system, or call **`stat()`** for any file in an SFS file system, and look at the **`blksize`** field.

Disk stripe (a.k.a. *file offset*)

The minimum amount of data required to generate or check parity.

SDA: 16 bytes * (number of SDA data disks)
= 128 bytes for an 8-disk SDA
= 256 bytes for a 16-disk SDA
= 385 bytes for a 24-disk SDA
= 512 bytes for a 32-disk SDA

DataVault: 512 bytes for any DataVault

Copyright © 1994 Thinking Machines Corporation

Disk stripe boundary Occurs every *n* bytes from byte 0, where *n* is the number of bytes in one disk stripe.

cmword SDA: 16 bytes
DataVault: 512 bytes

cmword boundary Occurs every *n* bytes from byte 0, where *n* is the number of bytes in one cmword.

3.2 Overview: Reading to and Writing from the CM-5

A basic read or write operation consists of two phases. For example, a read's two phases are, in order:

1. Raw data transfer: transferring data between the disks and the PNs' user memory. The transfer rate of this phase is usually[3] dominated by the speed of the SDA or DataVault disks. (For the SDA, the approximate speed is 1.5 Mbytes/second/disk.)

2. Data routing: Reordering the data in PN memory so that every PN has the correct data in the correct order for the application. The performance of this phase depends on several factors:

   - the number of processing nodes
   - the application's geometry, e.g., C* shape, CM Fortran array dimensions
   - the number of bytes per shape position or array element

I/O operations may happen while the initiating user process is context-switched out. For this reason, be especially careful when timing code that performs I/O; see Section 3.5.

3. If the number of PNs participating in the I/O operation is less than 32 multiplied by the number of SDA disk backplanes, the transfer rate of this phase is more greatly influenced by the number of PN backplanes.

Copyright © 1994 Thinking Machines Corporation

3.3 Obtaining Optimal File System Performance

For either the SFS or the CMFS, performance gets better as the number of bytes read or written gets larger. This general rule holds within the bounds of the specific rules given in Sections 3.3.1 through 3.3.3, below. If you need to perform relatively small transfers, see Section 3.4.

3.3.1 Parallel I/O Performance

The following performance rules apply to I/O operations performed from a CM-5 file system to the processing nodes (PNs) or vice versa:

Performance Rule 1

The most serious performance degradation occurs if an I/O operation does not meet both of the following conditions:

- Begin the read or write operation on a disk block boundary.
- Read or write integral disk blocks.

If both these conditions are satisfied, the operation can transfer the data over the quickest route — entirely over the CM-5 networks — and operate on the optimal amount of data — full blocks.

If both conditions cannot be met, the system must operate on at least one partial block, which degrades performance. For a partial-block write, for instance, the system must perform a read-modify-write operation, which requires the file server to perform a read prior to the write. This extra step causes a partial-block write to take twice as long as a full-block write.

Applications that perform both parallel and serial reads or writes to the same file are particularly susceptible to performance degradation caused by read-modify-write operations. There is little danger of landing between block boundaries, however, if you don't perform both parallel and serial reads or writes to the same file.

Copyright © 1994 Thinking Machines Corporation

Performance Rule 2

The next most serious performance degradation occurs if an I/O operation does not meet both of the following conditions:

- Begin the read or write operation on a disk stripe boundary.
- Read or write an integral number of disk stripes.

Write performance will be degraded because of slow parity generation if both of these conditions are not satisfied. (Parity generation requires an entire disk stripe's data. Parity calculations for a partial stripe take longer because the timesharing daemon must first read the rest of the stripe and send it to the PNs.) In addition, the Ethernet would be used for some of the I/O, which would also contribute to performance degradation.

3.3.2 Serial I/O Performance

When performing I/O from a CP or other serial computer to a CM-5 file system, or vice versa, start the transfer on a file system block boundary and transfer an integral number of file system blocks for best performance.

3.3.3 Specific Performance Hints for C* I/O

I/O performance may be significantly diminished if any of the following is true:

- The size specified via the third argument to **`CMFS_read_file_always`** or **`CMFS_write_file_always`** (or other I/O routines) is not a multiple of 4 bytes.
- The file position is not on a 4-byte boundary.
- The parallel data passed to **`CMFS_read_file_always`** or **`CMFS_write_file_always`** (or other I/O routines) is an address that is not on a 4-byte boundary (for example, when the pointer points to a member of a parallel structure).

Copyright © 1994 Thinking Machines Corporation

3.3.4 Specific Performance Hints for CM Fortran I/O

The following performance hints apply to programming in CM Fortran:

- Physical I/O — using either the **PHYSICAL** keyword with the **READ** and **WRITE** statements or the CM Fortran utility library's **_FMS** routines — is fastest, since it performs only the disk ↔ PN user memory phase of the I/O operation. Physical I/O is not appropriate for all applications, however; see the *CM Fortran Programming Guide*.
- The performance of implied **DO** loops is poor.

3.4 Performing Small Transfers

Because there is a minimum latency required to perform any I/O operation, you should try to ensure that the size of the data transfer is sufficient to minimize the effect of the latency. As a rule of thumb, if a data transfer must be less than 2−3 Mbytes, use the control processor's single disk file system rather than the DataVault or SDA.

3.5 Timing I/O Operations

To accurately time I/O operations, you must use UNIX wallclock time. For example:

```
/* get_time_of_day.c    /*

#include <stdio.h>
#include <sys/time.h>

void get_time_of_day_ (tv_sec, tv_usec)
long *tv_sec, *tv_usec;
{
    struct timeval tval;

    if (gettimeofday(&tval, NULL) != 0)
    {
         perror ("Error in gettimeofday");
```


Copyright © 1994 Thinking Machines Corporation

```
            exit(1);
        }
        *tv_sec = tval.tv_sec;
        *tv_usec = tval.tv_usec;
    }

    integer*4     tv_sec_1, tv_sec_2, tv_usec_1,
                  tv_usec_2

    call _get_time_of_day (tv_sec_1, tv_usec_1)
        /* I/O here */
    call _get_time_of_day (tv_sec_2, tv_usec_2)
```

Methods of timing that measure only the time a user process is running, such as CM or CMMD timer routines (**`CM_timer_`***xxx* or **`CMMD_node_timer_`***xxx*), the Fortran **`time()`** function, and the CM Fortran function **`ETIME()`**, cannot be used to measure CM-5 I/O performance accurately. If UNIX wallclock time is not used, timings for code that includes I/O will appear faster than they really are.

Copyright © 1994 Thinking Machines Corporation