Performance of the iPSC/860 Concurrent File System

Bill Nitzberg[1]

Report RND-92-020 December 1992

NAS Systems Development Branch
NAS Systems Division
NASA Ames Research Center
Mail Stop 258-6
Moffett Field, CA 94035-1000

Abstract

Typical scientific applications require vast amounts of processing power coupled with significant I/O capacity. Highly parallel computer systems can provide processing power at low cost, but tend to lack I/O capacity. By evaluating the performance and scalability of the Intel iPSC/860 Concurrent File System (CFS), we can get an idea of the current state of parallel I/O performance. I ran three types of tests on the iPSC/860 system at the Numerical Aerodynamic Simulation facility (NAS): broadcast, simulating initial data loading; partitioned, simulating reading and writing a one-dimensional decomposition; and interleaved, simulating reading and writing a two-dimensional decomposition.

The CFS at NAS can sustain up to 7 megabytes per second writing and 8 megabytes per second reading. However, due to the limited disk cache size, partitioned read performance sharply drops to less than 1 megabyte per second on 128 nodes. In addition, interleaved read and write performance show a similar drop in performance for small block sizes. Although the CFS can sustain 70-80% of peak I/O throughput, the I/O performance does not scale with the number of nodes.

Obtaining maximum performance may require significant programming effort: pre-allocating files, overlapping computation and I/O, using large block sizes, and limiting I/O parallelism. A better approach would be to attack the problem by either fixing the CFS (e.g., add more cache to the I/O nodes), or hiding its idiosyncracies (e.g., implement a parallel I/O library).

[1] Computer Sciences Corporation, NASA Contract NAS 2-12961, Moffett Field, CA 94035-1000

1.0 Introduction

Highly parallel computer systems have the potential of providing high performance at low cost. However, this potential cannot be realized unless the system components provide balanced, scalable performance. Typically, these systems have a peak megaflop rate and main memory capacity greater than traditional vector supercomputers. However, the I/O subsystem (secondary storage and network communication) is severely limited in its ability to keep up with the rest of the system. By examining the iPSC/860 Concurrent File System (CFS), we can get an idea of the current state of parallel I/O performance.

By examining the hardware and software designs, measuring the speed of file operations, and running a few detailed tests, I show:

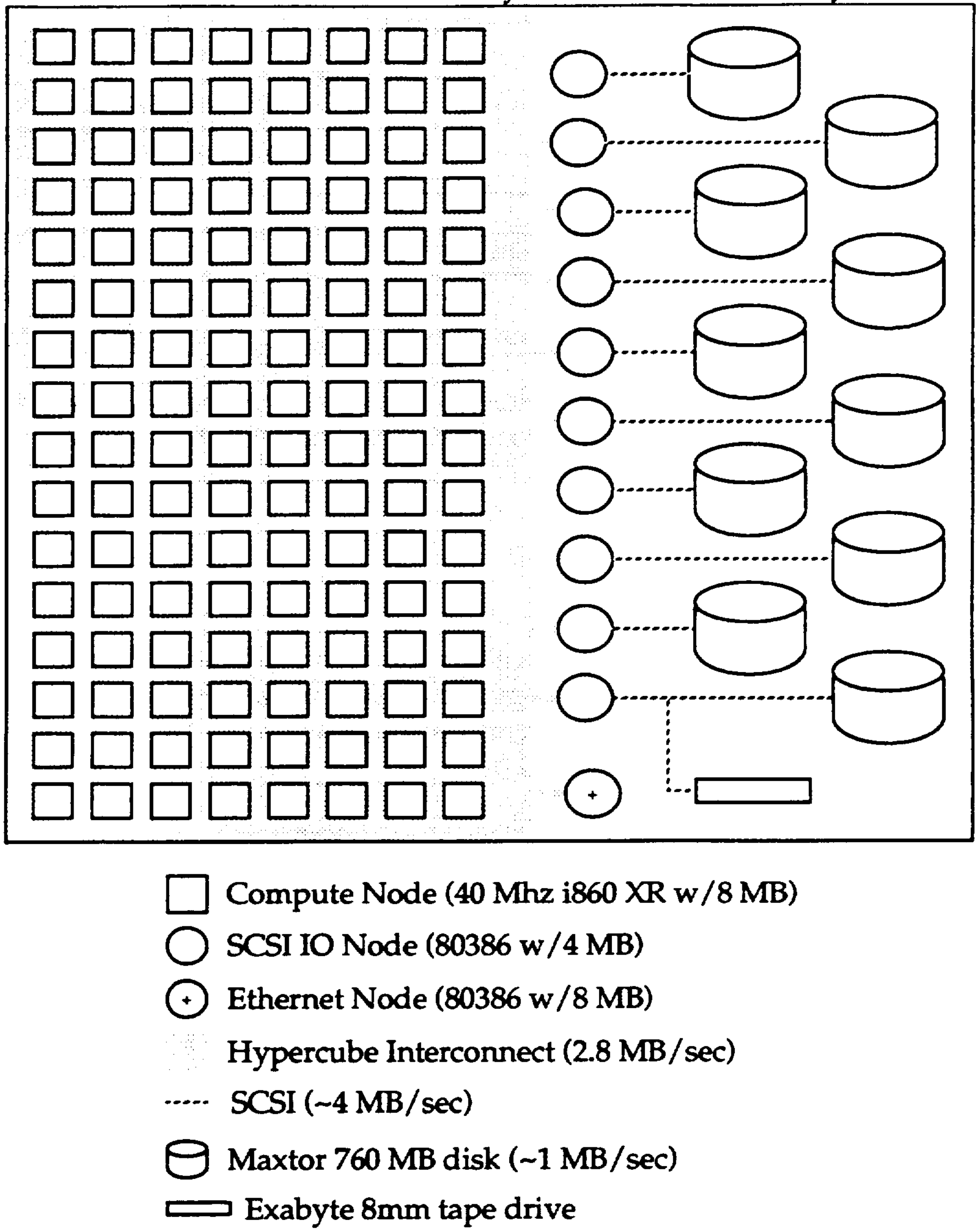
- the peak performance one can expect on a simple-minded application,
- why the CFS performs at the level it does,
- what performance one can expect from a larger system,
- how to improve performance on the current system.

I studied the performance of the CFS by running a set of tests designed to measure the peak I/O rate of a simple-minded user application. No attempt was made to measure average or multi-user performance, as this would be much more complicated, and the results would be too variable to be useful as a performance predictor. I looked at the hardware and software design of the CFS, and suggest several alternative hypothesis to explain unexpected performance results. In addition to examining these hypotheses, I looked at scalability: how performance scales with the number of compute nodes used. By varying the number of compute and I/O nodes used in the tests, I got an indication of how well the CFS will scale to a system with a greater number of nodes. Finally, I present a method of obtaining scalable performance and some programming hints to follow to obtain the best possible performance.

2.0 CFS Hardware

All of the performance tests were run on the iPSC/860 at the NAS facility. The iPSC/860 system is a hypercube-based MIMD parallel computer. The system at the NAS facility consists of 128 compute nodes, 10 I/O nodes, 1 Ethernet node, and an IBM PC-class front end computer. The CFS consists of the 10 I/O nodes, 10 SCSI disks, an Exabyte 8mm tape drive, and various library routines and servers.

FIGURE 1. The iPSC/860 System at the NAS Facility


Compute Node (40 Mhz i860 XR w/8 MB)
SCSI IO Node (80386 w/4 MB)
Ethernet Node (80386 w/8 MB)
Hypercube Interconnect (2.8 MB/sec)
SCSI (~4 MB/sec)
Maxtor 760 MB disk (~1 MB/sec)
Exabyte 8mm tape drive

Data transferring from the CFS to the compute nodes must travel from the disk, across a SCSI channel, to an I/O node. From the I/O node, the data must travel through the hypercube network to the destination compute node. The peak rate of the I/O system (also the peak rate of the CFS) is limited by the slowest link in the data path:

TABLE 1. Throughput Speeds

Device	Mbytes/sec
Hard Disk (Maxtor 8760S, 760 MB)	1
SCSI	4
I/O Node memory bandwidth (16 Mhz 80386)	64
Hypercube Interconnect (DCN)	2.8
Compute Node memory bandwidth (40 Mhz i860 XR)	160

In this case, the slowest path is the disk itself, with an estimated throughput of approximately 1 megabyte/second. This implies that the peak throughput rate that the hardware should sustain is 10 megabytes/second.

3.0 CFS Software

The software is divided into four parts: node libraries linked into user applications, NX operating system subroutines which are replicated on every compute node, disk block servers running on each I/O node, and a name server running on one I/O node.

A user application running on the compute nodes can access data on the CFS by first performing an `open()` system call. The call causes a request to be sent to the name server process which converts the file name into an absolute disk address for the file header information. This header, containing pointers to the data blocks on disk, is transferred to the compute node. Next, a `read()` or `write()` call will be converted into a series of requests to the disk block servers. For each disk block (4 kilobytes), the absolute disk address is looked up in the file header (residing locally at the compute node), and a request is sent to the disk block server on the appropriate I/O node. If more than 4 kilobytes are read/written in one call, the CFS library routines perform a series of requests, one at a time. (There is an asynchronous mode, but—due to a bug in the CFS software—it allows at most two requests to be performed at a time.)

The CFS uses striping and caching to improve performance. When a file is created (allocated), disk blocks are spread across all the disks in the system. Ideally, the file blocks would be placed one block per disk, in a round-robin fashion. This means that given N disks, a single I/O operation of N blocks would use all of the disks. Caching is done both for reading and writing. Each 4 megabyte I/O node reserves 1 megabyte for caching disk blocks. A read request is translated into a 32 kilobyte (8 block) request to the disk. A write request is not written to disk until necessary. Writes to contiguous disk blocks are grouped together, if possible, in 32 kilobyte pieces. The disk block server reads and writes 32 kilobytes at a time to obtain maximum throughput to the disks. Using 4 kilobyte blocks reduces throughput by a factor of three. However, when sending the data over the hypercube interconnect, data is sent in 4 kilobyte blocks to reduce contention on the network.

3.1 Peak Software Rates

The performance of the system was examined in the following manner. System times were measured for:

- I/O node memory bandwidth,
- compute node memory bandwidth,
- requesting and sending 4 kilobytes worth of data through the hypercube interconnect,
- reading and writing a single disk from a node.

TABLE 2. Measured Software Rates

Operation	MB/sec
I/O Node Memory Bandwidth (using `memset()`)	11.5
Compute Node Memory Bandwidth (using `memset()`)	48.7
Hypercube Network (Request/Reply with 4 kilobytes)	
nearest neighbor	2.2
across the whole machine	2.0
Disk to Compute Node (reading or writing)	0.82

These measurements imply that the maximum sustained I/O throughput of the CFS is 8.2 megabytes/second (= 10 disks * 0.82 MB/sec per disk).

4.0 Performance Tests

Five types of tests were performed: broadcast reading, partitioned reading and writing, and interleaved reading and writing. Broadcast reading simulates loading an initial data set onto every node. The partitioned tests simulate loading and saving a one dimensional decomposition of data. The interleaved tests simulate loading and saving a two dimensional decomposition of data. A further description of partitioned and interleaved is given in Section 5.2 on page 9. A more complicated layout of data, such as a three dimensional decomposition, is beyond the scope of this paper.

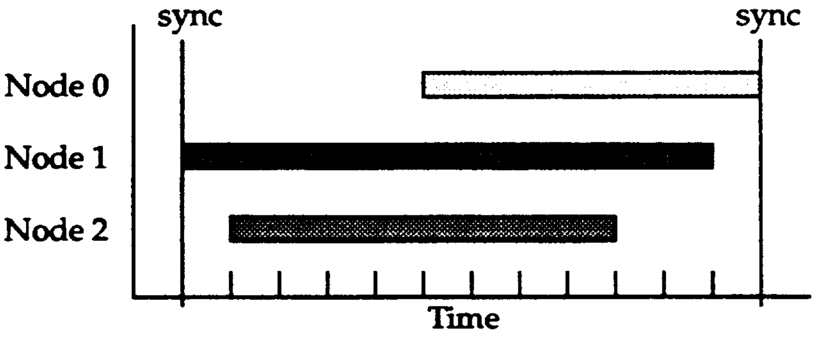
The testing parameters were varied to measure scalability and the effects of interleave size. I measured scalability by performing each test using 1, 2, 4, 8, 16, 32, 64, and 128 nodes. Only powers of two were used, as the iPSC/860 requires applications to run on a power of two number of nodes. I measured the effects of interleaving size by varying the block size read/written in a single file system call from 4 kilobytes to 1 megabyte. Smaller block sizes than 4 kilobytes, although common in user applications, result in very poor performance. Since the project goal was to determine attainable peak performance on simple applications, performance using small block sizes was not measured.

Each test was performed as follows:

```
Globally Synchronize all Nodes
Start Timing
Open the File
Perform the I/O Test
Close the File
Globally Synchronize all Nodes
Stop Timing
```

Wall clock time was independently measured on each node. The time reported for a test is the maximum of the times across all nodes. The global synchronization ensures that this maximum corresponds to the total wall clock time.

FIGURE 2. Example Runtime Chart



Consider the example in Figure 2. The shaded areas represent the individually measured running times. Without the global synchronization, the maximum running time would be 11 time units (node 1). However, the total wall clock time required to complete the job on all nodes is 12 time units. The global synchronization forces wall clock time to be measured from the beginning of the first node's execution to the end of the last node's execution. Otherwise, a node with maximum individual running time which finished before another node would cause the running time to be under reported.

I measured the overhead of the operations given above to ensure that the results reflect throughput, and not overhead.

TABLE 3. Overhead Measurements

Operation	Seconds
Globally Synchronize all Nodes	0.000002
Open/Read a Byte/Close	0.072
Open/Write a Byte/Close	0.042

The partitioned and interleaved tests were all performed using a 128 megabyte file. This size was chosen to approximate user application file size (20-160 megabytes), minimize the affects of buffering (and measure sustained performance), and reduce overhead to an acceptable level. All of these tests required a minimum of 15 seconds to complete (0.5% overhead). The fastest test (a broadcast read test) completed in 3.5 seconds (2% overhead). Due to a bug in the CFS software (which corrupted randomly written files), the files used in all tests had to be preallocated. By reserving space on the CFS before performing each test, this bug was avoided.

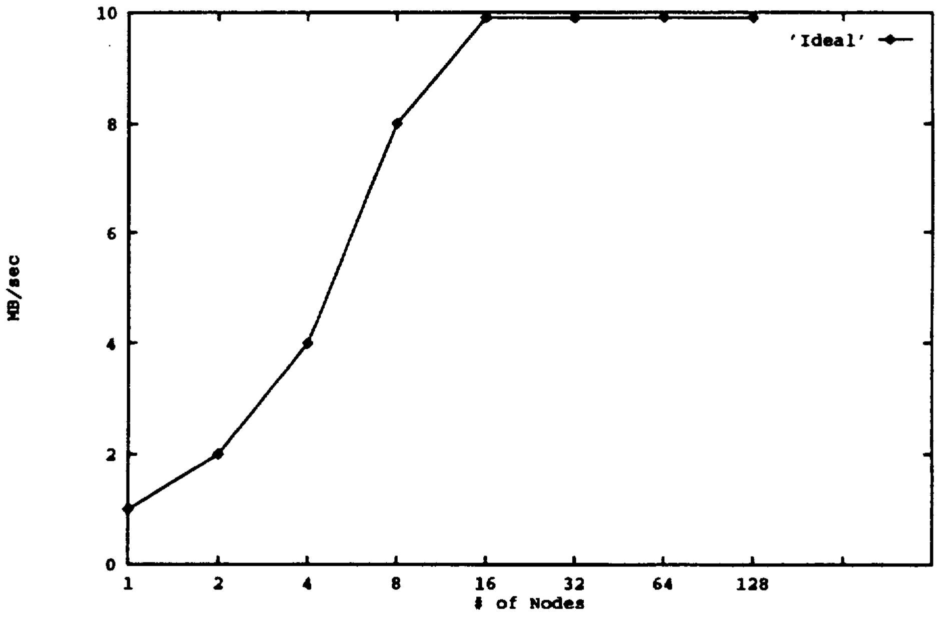
The tests were run on a semi-dedicated machine by using all 128 nodes at night via NQS (a batch job scheduling facility). Although no other jobs could run on the compute nodes (as every test reserved all 128 nodes), the system was still connected to the network; users could login to the front-end and access files on the CFS via the Ethernet I/O node. Running the tests at night minimized these activities. Further, each test was repeated three times. The results reflect the best of the three trials. Selected tests were re-run on a fully dedicated system (with no users on the iPSC/860 or the front-end, and the system isolated from the network) to verify that the NQS runs were true reflections of the performance of the CFS. The performance difference between the fully dedicated and semi-dedicated system runs was insignificant. The iPSC/860 was running the NX operating system, version 3.3.1, 3/92 update.

5.0 Performance Results

The performance results should be viewed in the light that timings varied by more than 20% from one run to the next. An interleaved write test using 32 nodes, a 32 megabyte file, and a 32 kilobyte block size was repeated six times on a dedicated machine. The performance ranged from 5.5 to 7.6 seconds (4.2 to 5.8 megabytes/second), which is more than a 20% difference. Small timing variations that occur can be magnified by the asynchronous nature of the iPSC/860 to cause large overall timing variations. By reporting only the best of the three runs for each test, the observed affects of this variation are reduced.

Results are reported in megabytes per second, and are plotted against the number of nodes used in the test (Figure 4 - Figure 14). In general, the performance of a perfectly scaling I/O system should increase linearly with the number of nodes up to the peak I/O rate. Once the peak rate is reached, the performance should remain at the peak rate up to the maximum number of nodes in the system (Figure 3).

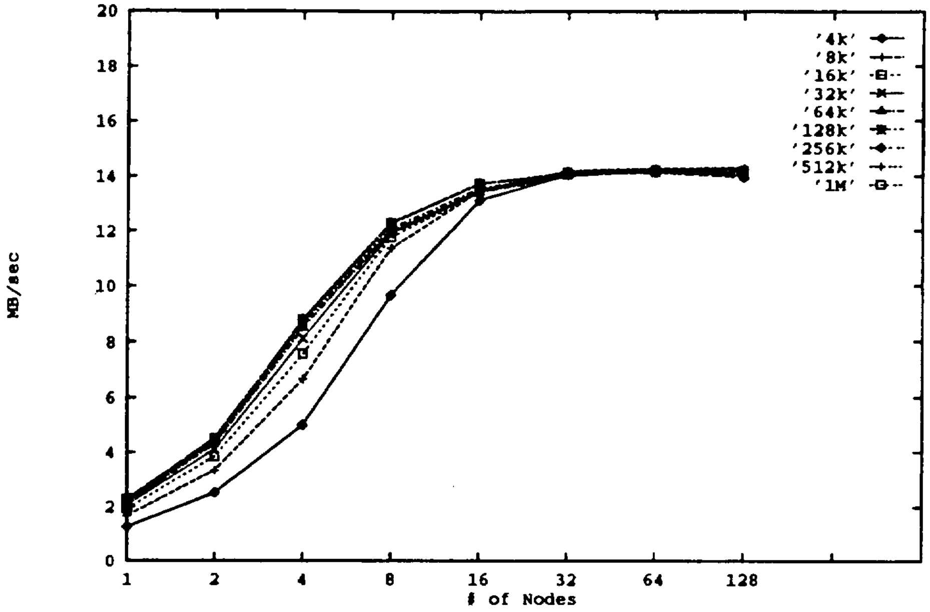
FIGURE 3. Ideal Performance Curve



5.1 Broadcast Read

Broadcast reading simulates loading an initial data set onto every node. Each node reads the same file in its entirety—the file is "broadcast" from the disk to every node in the system. Since there are only 8 megabytes of memory on each node, the file size used for this test was 8 megabytes. Broadcast read performance (Figure 4) grows proportionally with the number of compute nodes (as expected), and peaks at approximately 14 MB/sec. Block size does not affect performance, although the 4k block size performs slightly worse than the others.

FIGURE 4. Broadcast Read (8 MB file)



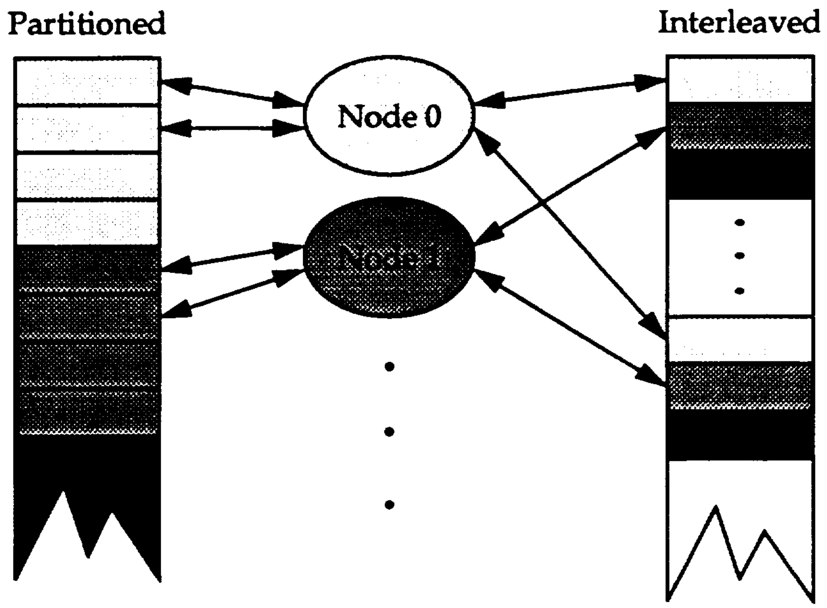
Note that peak rate (14 MB/sec) is higher than the stated theoretical peak rate for the installed hardware (10 MB/sec). This increased throughput is the result of caching on the I/O nodes, as the same data is being read by all nodes.

In this example, disk bandwidth is the limiting factor. Broadcasting data from a file on disk can be performed much faster by reading the data into a few nodes' memories, then performing a tree broadcast using the hypercube interconnect of the iPSC/860 system. A better estimate for peak throughput in this case would be approximately 60 MB/sec; eight nodes read the file, then perform a tree broadcast using the hypercube interconnect.

5.2 Partitioned vs. Interleaved

A partitioned file corresponds to a one dimensional decomposition, while an interleaved file corresponds to a two dimensional decomposition. In the partitioned case, the block size does not significantly affect performance. A larger block size will reduce the overhead of repeatedly calling the system I/O routines. In the interleaved case, however, the block size determines the amount of interleaving (the size of the rapidly changing dimension of the two dimensional decomposition). Figure 5 is a graphic representation of the two file formats.

FIGURE 5. File Formats (Accessing the First 2 Blocks)



Both of these file formats were used since they both occur in practice, and are relatively easy to implement. Note that both formats are the same for the case when the file is accessed by a single node or when the number of nodes equals file size divided by block size.

5.3 Partitioned Tests

As with the broadcast read test, the partitioned results are not affected significantly by block size. For partitioned files, block size corresponds to I/O system call overhead. The larger the block size, the fewer I/O system calls must be performed to transfer the data. For this reason, the 4 kilobyte block size case performs slightly worse than the others.

The partitioned write test yields expected performance (Figure 6): scaling from 2 megabytes per second on one node to 6-7 megabytes per second on 16 nodes, then leveling off at this rate (dropping only slightly) through 128 nodes. The slight drop in performance above 16 nodes is most likely due to synchronization and contention. The performance curve for this test is nearly ideal; the sustained partitioned write performance of the iPSC/860 is 6-7 megabytes per second.

The partitioned read test (Figure 7), however, exhibits a surprising sharp drop in performance. The performance scales nicely from 2 megabytes per second on one node through 8 megabytes per second on 16 nodes, but then drops to below 1 megabyte per second on 64 and 128 nodes. Not only does the partitioned read test not scale, but the performance is worse at 128 nodes than it is on a single node. This performance anomaly is due to a poor caching mechanism. Disk caching is examined in Section 6.3 on page 15.

FIGURE 6. Partitioned Write (128 MB file)

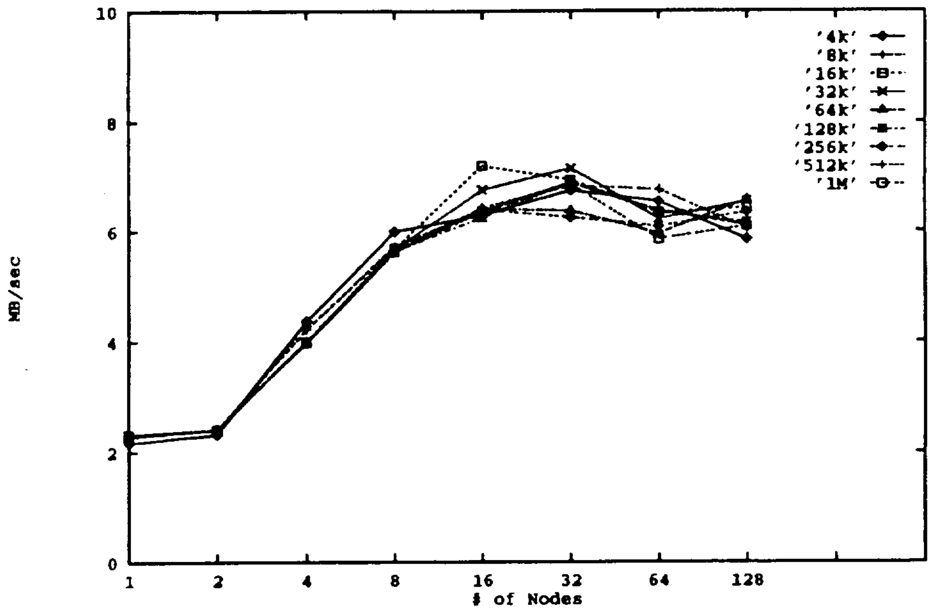
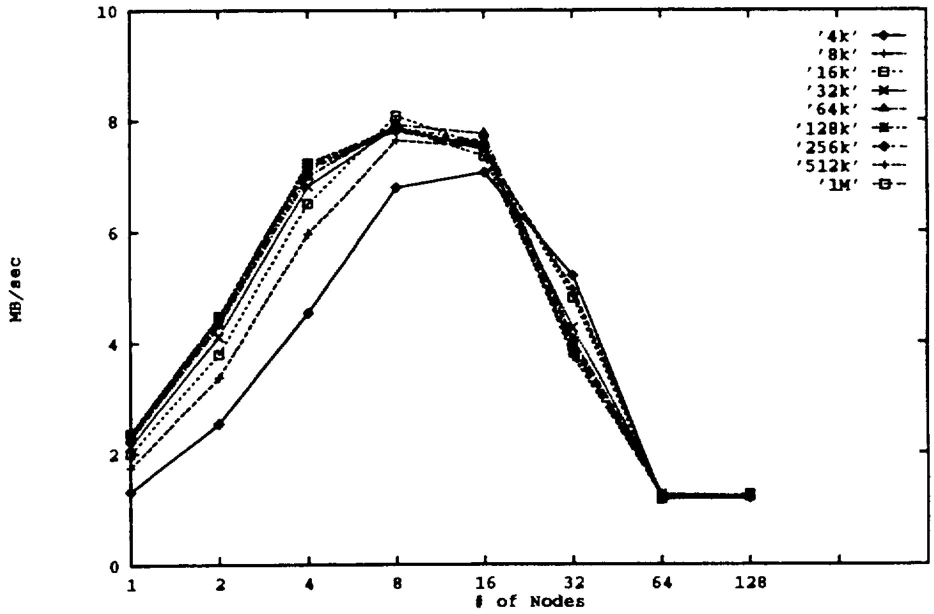


FIGURE 7. Partitioned Read (128 MB file)



5.4 Interleaved Tests

As expected, the performance results for the interleaved tests are much more dependent on block size, since block size determines the amount of interleaving.

The interleaved write tests (Figure 8) perform as expected for large block sizes (above 128 kilobytes). For large block sizes, the performance scales from about 2 megabytes per second on a single node to about 5-6 megabytes per second on 16 through 128 nodes. For block sizes of 128 kilobytes and below, the performance curve is more bell shaped, peaking at 5-7 megabytes per second on 16 nodes, then falling to below 4 megabytes per second on 128 nodes.

The interleaved read tests (Figure 9) perform identically to the partitioned read tests through 8 nodes (increasing from about 2 megabytes per second to about 7-8 megabytes per second). When run on more than 8 nodes, the performance degrades. Except for the 32 and 64 kilobyte block sizes, the larger the block size, the worse the performance on more than 8 nodes. This is expected (given the partitioned test results) as the 1 megabyte block size, 128 node interleaved read and partitioned read tests are identical. The 512 kilobyte block size tests, although not identical, are very similar, etc. It is unclear why the 64 kilobyte block size performed so much better than the others.

FIGURE 8. Interleaved Write (128 MB file)

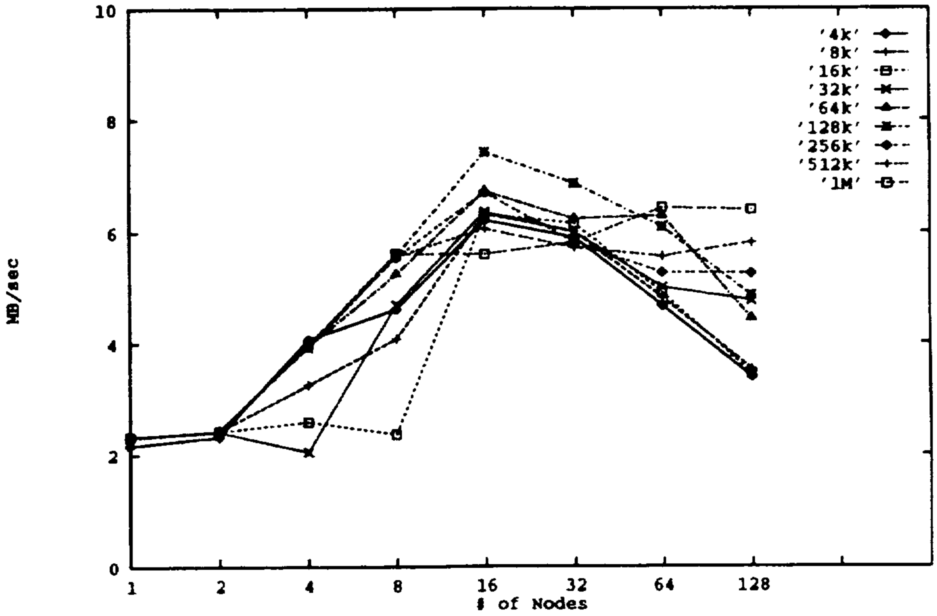
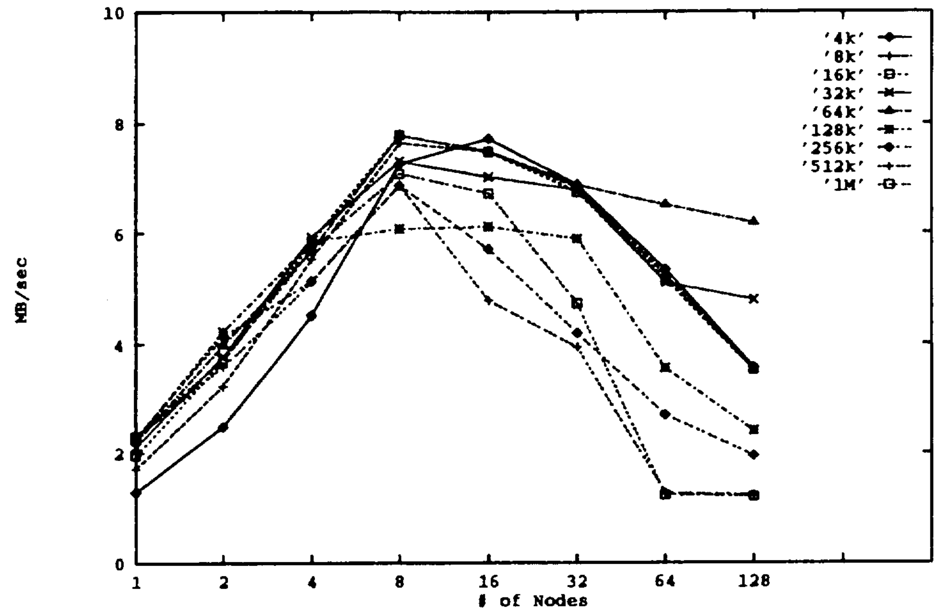


FIGURE 9. Interleaved Read (128 MB file)



6.0 Investigating the Performance Drop

On all of the tests performed, the CFS scales almost ideally from 2 megabytes per second using a single node through 6-8 megabytes per second using 16 nodes. However, except for the partitioned write tests, the performance drops (sharply) on more than 16 nodes. It is curious that major papers on the performance of the CFS [4, 6] do not contain performance results for more than 16 nodes. I ran some further tests to determine the cause of the performance drop.

6.1 Latency and Contention

Network latency and network contention were investigated as a source for the performance anomaly. Although the compute nodes in the iPSC/860 are fully connected in a hypercube, each I/O node is connected over a single link to one compute node. It is standard manufacturing practice to anchor (connect) the I/O nodes to the low numbered compute nodes. In our system, the I/O nodes are connected to compute nodes 2, 6, 10, 14, 18, 26, 30, 34, 38, and 42. Further, all of the CFS tests allocated nodes starting from node 0. So, a 16 node test would use nodes 0-15, while a 64 node test would use nodes 0-63. The placement of the I/O nodes within the system combined with the placement of the CFS tests among the compute nodes may have contributed to the performance drop.

First, I investigated latency as a possible cause. Since the speed at which disk requests can be made decreases as distance increases (see Table 2 on page 5), I re-ran selected 16 node tests, varying the location of the 16 compute nodes used within the system. If latency is a major factor in performance, one would expect the tests run on low numbered nodes (close to the I/O nodes) to perform the best, and the tests run on high numbered nodes (far from the I/O nodes) to perform the worst. However, performance was not affected by location, implying that internal network latency is not causing the problem.

Next, I investigated contention. If multiple messages are sent across a network link simultaneously, they must contend for access to the link; typically, the message transfers are serialized. For example, if a network link runs at 1 megabyte per second, and two messages, each 1 megabyte in size, are sent over the link, it will take at least two seconds to complete the transfer. The iPSC/860 uses a hypercube routing algorithm that routes messages from lowest dimension to highest dimension. A message from node 63 to node 2 must traverse links 63-31, 31-15, 15-7, 7-3, and 3-2. To go from node 63 to node 6, the links 63-31, 31-15, 15-7, and 7-6 must be traversed. Therefore, two disk requests from node 63 to the I/O nodes directly connected to nodes 2 and 6 must traverse three identical links. Similar contention for links can occur throughout the system. The contention for these links increases with the number of nodes used.

However, the CFS system sends requests to the I/O nodes sequentially (for synchronous I/O). So each compute node has at most one outstanding request at a time. A bug in the CFS system limits outstanding asynchronous I/O requests to two—or a total of 256 for the whole system. This many requests should not be enough to cause significant contention. I reconfigured the iPSC/860 hardware to confirm that the placement of I/O nodes within the system was not affecting performance. The I/O nodes were re-anchored, evenly spaced every 12 nodes within the system. Again selected tests were re-run on the system, and again, the performance remained unchanged.

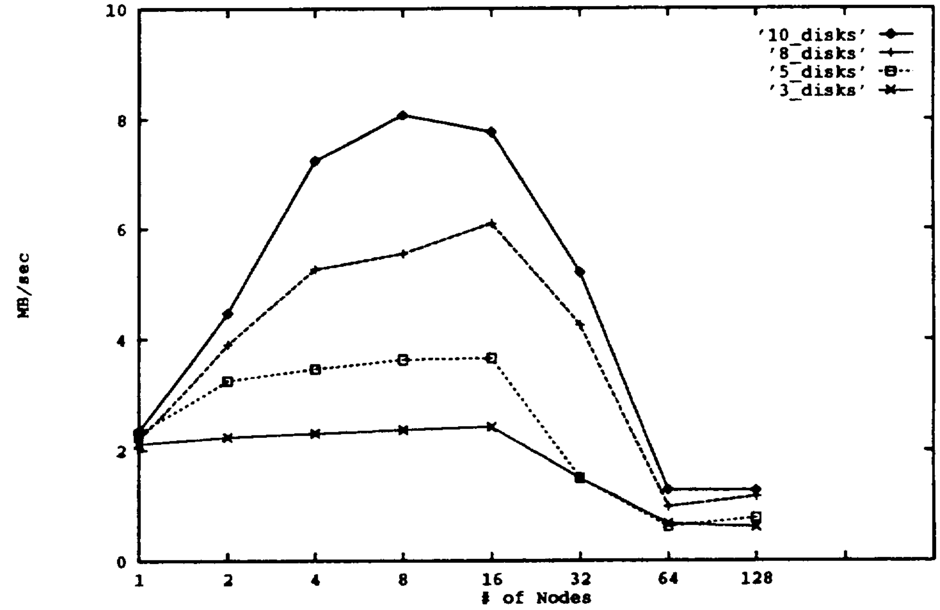
The test results indicate that neither latency nor contention are the major cause of the performance drop. It would be unlikely for either to have played a major role in the performance drop, as the I/O performance is substantially lower than the network performance. In general, the effects of latency and contention are most noticeable when the measured performance is close to the sustainable network rate.

6.2 Larger System Simulation

The most straightforward method of investigating the scalability of a system is to scale the system and measure its performance. With the iPSC/860, this is impossible beyond 128 compute nodes due to hardware limitations. I simulate a larger system by reducing the number of I/O nodes. A system with 128 compute nodes and 5 I/O nodes should have similar I/O performance to a system with 256 compute nodes and 10 I/O nodes. Although this is not ideal, it should give some insight into the behavior of the system.

The I/O system allows one to restrict file operations to a subset of available disks via `restrictvol()`. Using this mechanism, the number of disks used was reduced to 8, 5, and 3, and the partitioned read tests were re-run (Figure 10). One would expect the point at which the performance drops to vary with the number of disks used. However, the performance always dropped off at 16 nodes.

FIGURE 10. Partitioned Read (Varying # of Disks)



The performance on 64 and 128 nodes is almost identical for every number of disks tested. This implies that adding more disks or I/O nodes to the system would not improve the throughput on a large system. These unexpected results lead to the conclusion that the performance drop is a global property of the I/O system, such as caching.

6.3 Investigating Disk Cache Behavior

The most likely explanation for the drop in read performance is thrashing. The I/O system on the iPSC/860 uses disk block caching and pre-fetching to improve performance. However, the combination of caching and pre-fetching leads to greatly reduced performance in certain cases (such as in the partitioned read tests).

Caching improves performance by exploiting data re-use. If a disk block is read multiple times, it can be read from disk once (an expensive operation), stored in cache, and then read from cache for succeeding operations. However, since cache size is limited, typically, only recently accessed blocks are stored in cache. The partitioned and interleaved reading and writing tests did not re-use any data. In fact, for reading and writing large solution files, it is doubtful that caching would ever be used.

However, the I/O system also uses pre-fetching to improve performance. Pre-fetching depends on locality of reference and requires caching to be effective. When a disk block is accessed, it and several successive blocks are read into cache. It is typically more efficient to read multiple blocks from disk in a single operation than to read the same blocks, one disk operation at a time. When the successive blocks are accessed, they can simply be returned from cache. If, however, the successive blocks are not accessed, or are purged from cache before they are accessed, then the extra disk I/O to pre-fetch them was wasted. All of the tests could benefit from pre-fetching; however, the limited size of the disk cache combined with the amount of pre-fetching can cause nearly all of the pre-fetching to be wasted.

The 4 megabyte I/O nodes on our system each reserve approximately 1 megabyte (space for 250 disk blocks) for caching. Pre-fetching is performed 8 blocks at a time, so there is space to store pre-fetches from 250/8 = 31.25 different areas of a file on each I/O node.

Consider a single node sequentially reading a file which is uniformly stripped across all 10 I/O nodes. Reading the first 10 blocks will cause 10 pre-fetch operations, one on each I/O node, causing total of 80 blocks to be transferred from disk and placed into cache. The next 70 blocks can be read from cache. Reading the following 10 blocks will cause a second set of pre-fetch operations. At this point, it is no longer necessary to keep the first set of 80 blocks in the cache. The second set can use the same space. In this case, pre-fetching works nicely, and each I/O node requires 8 blocks of cache to store pre-fetched data.

For the partitioned read test, each node reads a different area of the file. Therefore, each I/O node requires space for 8 cache blocks for each compute node used. With 16 nodes, the cache on each I/O node must be at least 16 * 8 = 128 blocks. With 32 or more nodes, however, there must be at least 32 * 8 = 256 blocks. Since there is only 250 blocks available, the pre-fetch data for 32 or more nodes will not fit in cache.

Now consider running the partitioned read test using 64 compute nodes. Restrict attention to one I/O node for simplicity. The first 31 nodes read a block, causing 31 pre-fetch operations, using 248 of the 250 cache blocks available. When the next 31 nodes read a block, they also cause pre-fetch operations, but since there is no more free space in cache, each pre-fetch must purge the data from a previous pre-fetch operation. Most likely, data is purged before anything but the first block is read by the requesting node. This continues for the entire file. For every read operation, 8 blocks are pre-fetched, 1 block is returned, and the other 7 are purged from cache before they can be read. This should result in an 8X slowdown from 16 to 64 nodes, but no performance difference between 64 and 128 nodes—exactly what was measured (Figure 7).

Interleaved reading exhibits this problem too. For large block sizes, the results are the same as for partitioned. As the block size is decreased, performance improves. A smaller block size allows better use of cache. As the block size decreases, the reads are closer together in the file, and there is more chance that each read will be able to use pre-fetched data. Since there is no way to use pre-fetching for writes, the writing does not suffer from this problem.

Note that a second performance drop can be seen for both the interleaved read and interleaved write tests. Performance drops for small block sizes (4, 8, and 16 kilobytes) above 16 nodes. Further investigation is required to determine the cause of this performance drop.

7.0 Limiting I/O Parallelism

It appears that the significant performance drop results from numerous independent I/O requests causing pre-fetches which overflow the disk cache. One solution to this problem is to restrict the number of independent I/O requests, i.e. limit the amount of I/O parallelism. Since the peak performance is observed on 16 nodes, I modified the partitioned and interleaved tests to arrange nodes into 16 logical groups, then added synchronization code to ensure that at most one I/O operation per group was active at a time. Finally, the partitioned and interleaved tests were rerun.

The partitioned write performance (Figure 11) was basically unchanged. Variation in test results from one run to the next accounts for the minor differences between the grouped and non-grouped partitioned write results. The partitioned read results (Figure 12) are dramatically improved. The peak at 8 megabytes per second drops to only 7 megabytes per second on 128 nodes. It is likely that this method would scale above 128 nodes. The slight drop (from 8 to 7 megabytes per second) can be attributed to synchronization overhead.

FIGURE 11. Grouped Partitioned Write (128 MB file)

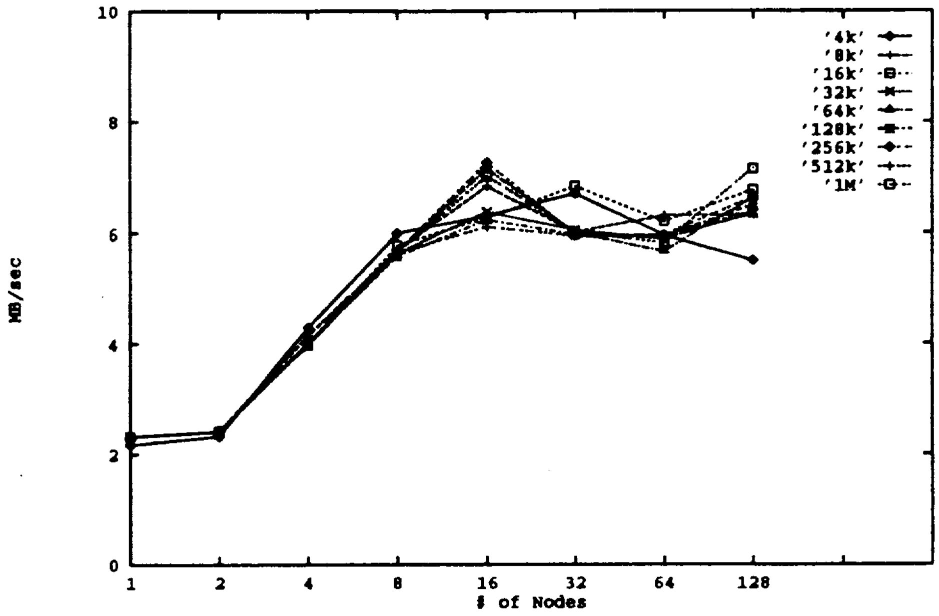
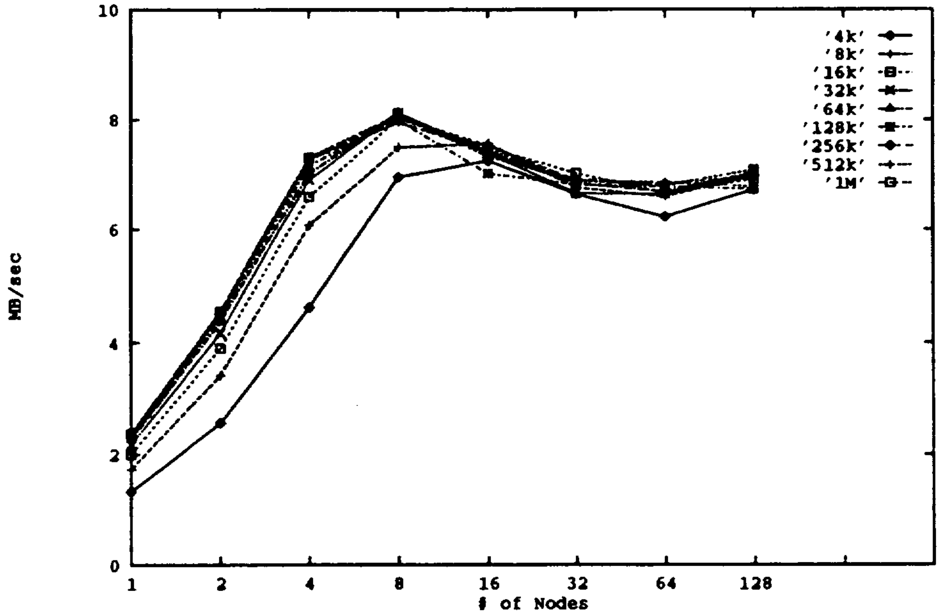


FIGURE 12. Grouped Partitioned Read (128 MB file)



The interleaved tests showed less dramatic improvement. Again, the write performance (Figure 13) was basically unchanged between grouped and ungrouped. For reading (Figure 14), the larger block sizes, greater than 32 kilobytes, had noticeably improved performance. As noted before, as block size increases, so does the similarity to the partitioned algorithm.

FIGURE 13. Grouped Interleaved Write (128 MB file)

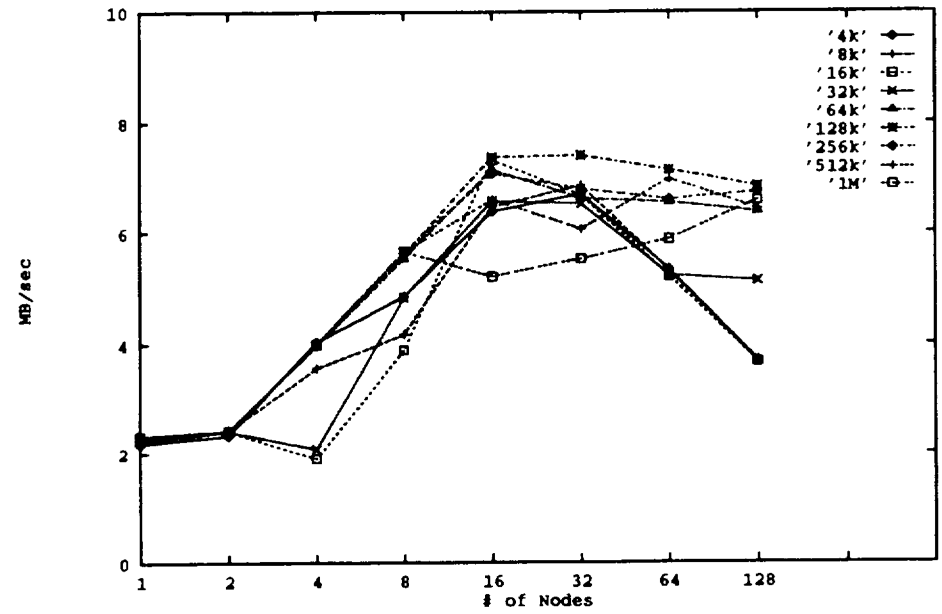
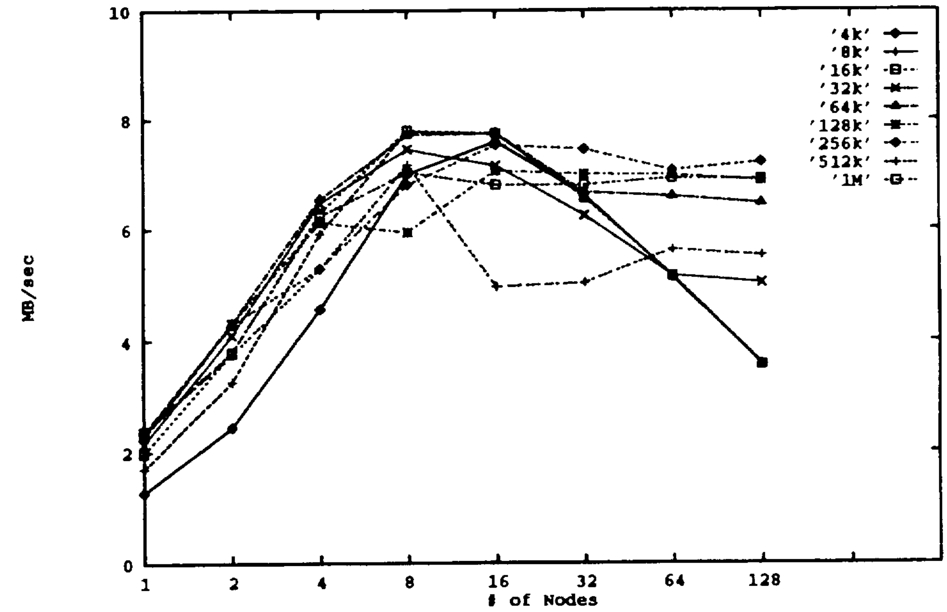


FIGURE 14. Grouped Interleaved Read (128 MB file)



Overall, grouping greatly improves performance when using more than 16 nodes. According to [8], the maximum performance is obtained by grouping nodes as above, but also restricting each group to using its own disk. However, according to [4] this produces the worst performance. In addition, this has the major drawback that multiple independent files must be used and managed.

8.0 Programming Hints

The following is a list of programming hints to maximize I/O performance. Some are generally applicable, while others are specific to the iPSC/860.

8.1 Pre-allocate files

A bug in the CFS system requires that files which are randomly written be preallocated. In general, this is a good idea in any high performance computer environment, not only for the speed improvement, but also to reserve the space. It is wasteful to run an eight hour job, only to have it die due to lack of space while trying to write the results.

In addition, on the iPSC/860, files created on a freshly made CFS will provide increased performance, typically exhibiting performance on read or write of 7-8 megabytes per second instead of 6-7 megabytes per second peak. With use, creating and deleting files, the CFS becomes fragmented, and no longer stripes files efficiently.

8.2 Overlap computation & I/O

Another general rule when trying to improve I/O speed in any system is to overlap as much I/O as possible with computation. On the iPSC/860, this can be accomplished by performing asynchronous I/O. Since there was no computation in the CFS tests run, asynchronous I/O was not tested.

8.3 Perform I/O operations with large block sizes

All of the tests performed for this report used a block size of 4 kilobytes or larger. Smaller block sizes gave rotten performance—below 1 megabyte/second.

For a machine like the iPSC/860 which has a large performance disparity between disk I/O and network communication rates, it is preferable to gather the data on the compute nodes into at least 4 kilobyte blocks, then write it to disk. This data rearrangement operation may be somewhat expensive, but it is far less expensive than simply writing the data in very small chunks.

8.4 Limit I/O parallelism

Finally, maximum performance is obtained by limiting the amount of I/O parallelism through grouping. At most 16 nodes should be performing an I/O operation at once. This will give peak throughput to the I/O system.

An alternative grouping is to pair each I/O node with a subset of the compute nodes. For example, on our iPSC/860, every 12 compute nodes would write to a single I/O node. However, this is more complicated to implement, inhibits file sharing, and limits the maximum throughput obtainable to any individual file.

9.0 Recommendations

9.1 Larger Block Cache

The most likely cause of the performance drop is the disk cache. A simple work-around to the problem would be to increase the cache size. This is not a scalable solution, as the cache size on every I/O node would need to be increased, and the performance drop would still occur—although it might occur when running on more than 128 nodes.

If possible, the CFS tests should be re-run on a machine with 8 megabyte I/O nodes instead of the 4 megabyte I/O nodes on the system at the NAS facility. If the limited size of the disk cache is the cause of the performance drop, increasing the cache by a factor of 5 (by expanding cache from 1 megabyte to 5 megabytes) should nearly eliminate the problem on the current system.

9.2 Parallel I/O Library

Highly parallel systems are difficult to program, and in general, obtaining peak performance from the I/O system will require a significant amount of effort. Further, this effort must be repeated for each highly parallel system one wishes to use. A library of global I/O routines would not only allow portable programs to be written with minimal effort, but it would also allow programmers to concentrate on algorithm development and execution speed instead of I/O performance.

The library should include routines for reading and distributing, as well as collecting and writing, one, two, and three dimensional decompositions of arrays. On the iPSC/860, the library could incorporate both the grouping of nodes and the "transpose" operation necessary to ensure that block sizes are above 4 kilobytes. This would greatly increase throughput for the average user.

10.0 Acknowledgments

Thanks to: Chris Kuszmaul for discussing anomalous CFS behavior with me on numerous occasions; Russell Carter for editing and encouraging me to finish this report before it was completely obsolete; and Art Lazanoff and William Smith for helping me re-anchor the I/O nodes on the

iPSC/860 (and especially for forcing me to label them with their original positions).

11.0 Summary

By evaluating the performance and scalability of the Intel iPSC/860 Concurrent File System (CFS), I have examined the current state of parallel I/O performance. I ran three types of tests on the iPSC/860 system at the Numerical Aerodynamic Simulation facility (NAS): broadcast, simulating initial data loading; partitioned, simulating reading and writing a one-dimensional decomposition; and interleaved, simulating reading and writing a two-dimensional decomposition. The CFS at NAS can sustain up to 7 megabytes per second writing and 8 megabytes per second reading. However, due to the limited disk cache size, partitioned read performance sharply drops to less than 1 megabyte per second on 128 nodes. In addition, interleaved read and write performance show a similar drop in performance for small block sizes. Although the CFS can sustain 70-80% of peak I/O throughput, the I/O performance does not scale with the number of nodes. Finally, obtaining maximum performance may require significant programming effort: pre-allocating files, overlapping computation and I/O, using large block sizes, and limiting I/O parallelism. A better approach would be to attack the problem by either fixing the CFS (e.g. add more cache to the I/O nodes), or hiding its idiosyncracies (e.g. implement a parallel I/O library).

References

[1] Ashbury, Raymond K., and David S. Scott, "Fortran I/O on the iPSC/2: Is There Read After Write?", Proc. 4th Conf. on Hypercube Concurrent Computers and Applications, Monterey, March 1989, pp. 129-132.

[2] Crockett, Tom W., "File Concepts for Parallel I/O", Proc. Supercomputing '89, Reno, Nevada, November 1989, pp. 574-479.

[3] Eckland, Denise, Intel Supercomputer Systems Division, personal communication, June 1992.

[4] French, James C., Terrence W. Pratt, and Mriganka Das, "Performance Measurement of a Parallel Input/Output System for the Intel iPSC/2 Hypercube", Proc. of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, San Diego, CA, May 21-24, 1991, pp. 178-187.

[5] Lou, Zhong C., "A Summary of CFS I/O Tests", NAS Systems Division technical report RNR-90-020, NASA Ames Research Center, MS T045-1, Moffett Field, CA, October 29, 1990.

[6] Pierce, Paul, "A Concurrent File System for a Highly Parallel Mass Storage Subsystem", Proc. 4th Conf. on Hypercube Concurrent Computers and Applications, Monterey, March 1989, pp. 155-160.

[7] Ryan, James S., "Concurrent File System (CFS) I/O for CFD Applications", unknown publication status.

[8] Scott, David, Intel Supercomputer Systems Division, personal communication, June 1992.