

Parallel I/O from the User's Perspective

Jacob Gotwals Suresh Srinivas Shelby Yang
Department of Computer Science
Lindley Hall 215, Indiana University
Bloomington, IN, 47405
{jgotwals,ssriniva,yang}@cs.indiana.edu

Abstract

Parallel I/O systems are gaining popularity as a means for providing scalable high bandwidth I/O. In this paper we develop abstract models of parallel I/O systems and provide empirical results that show how I/O intensive applications interact with the elements of parallel I/O systems. The abstract models are useful for explaining parallel I/O performance to developers of applications requiring high performance I/O and the experimental results provide insight into the I/O performance of present generation parallel machines. We also identify optimizations for improving I/O performance, which should be useful for developers of applications and I/O libraries.

1 Introduction

Restricting a parallel machine to a single I/O device would cause I/O to become a bottleneck, since I/O power would be unable to scale with computational power. Therefore, many present-generation parallel machines provide a scalable parallel I/O subsystem consisting of several I/O nodes each connected to a high speed I/O device. This enables high bandwidth co-operative I/O *i.e.*, *parallel I/O* to be performed from the compute nodes [1, 2, 3]. Many large-scale parallel applications require the use of parallel I/O to achieve high performance [4].

In this paper, we design models and experiments to examine how parallel I/O systems perform and scale. We provide experimental results for parallel I/O scalability and performance on the Intel Paragon and the Thinking Machines CM-5. The models and results presented in this study should be useful for optimizing I/O performance in applications, configuring existing parallel I/O systems and designing new ones.

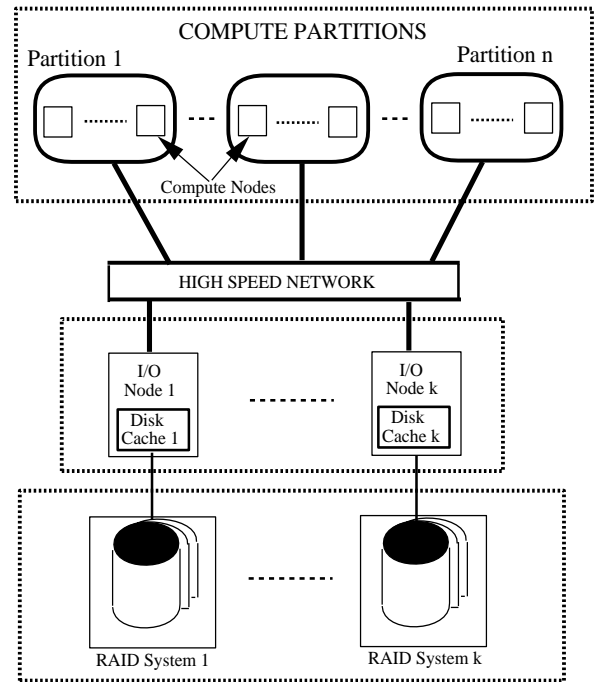


Figure 1: **ARCHITECTURAL ELEMENTS OF DISTRIBUTED MEMORY I/O SYSTEMS:** A variable number of I/O nodes share a scalable network with the compute nodes, allowing I/O power to be scaled with compute power. Each I/O node has a disk buffer and is connected to a disk device, which could be a RAID (Redundant Array of Inexpensive Disks) subsystem. The compute nodes in the parallel machine are grouped into compute partitions. The I/O nodes do not belong to any one compute partition but are shared across all of them.

2 Parallel I/O systems: A brief overview

Distributed memory parallel machines are converging towards a common architecture for parallel I/O

systems (see Figure 1). Some of the present generation parallel machines that provide such a system are the Intel Paragon, Thinking Machines CM-5 [1] and IBM SP-1 [5].

Both the Paragon and the CM-5 provide a special UNIX file system that gives applications high speed access to the parallel I/O system. These file systems can be used in the same way as normal UNIX file systems, but in addition, they are accessible through special system calls provided for parallel I/O. The system call interface for parallel I/O is similar to the interface for normal UNIX I/O, with a single extension. An *I/O mode*, associated with each file descriptor, coordinates the way in which the compute nodes access the file.

3 Performance and scalability of I/O in three models of parallel I/O systems

In this section we start with a simple model of a parallel I/O system, then successively refine the model, at each step discussing the relationship between the elements of the model and the way I/O performance scales under increasing I/O loads.

Let us start with a machine having p compute nodes and a parallel I/O system with one I/O node (see Figure 2). Assume that:

- An application, when run on the compute nodes, issues a parallel input or a parallel output request to the I/O system. The parallel I/O request consists of a system call executed by each compute node which causes a total of n bytes to be concurrently read or written (n/p bytes on each node).
- The time for I/O is the same on each node.
- We repeatedly run our application for increasing output sizes, and that we wait for the I/O system to clear after each run.

For such a parallel I/O operation we define:

- *Application I/O time* denoted by $Time_{app}(n)$ as the time the system call takes to move data between the application's buffer and the top level of the I/O system.
- *Actual I/O time* denoted by $Time_{actual}(n)$ as the actual time for the I/O operation to complete *i.e.*, the time it takes for data to go all the way between the application and the disk.

These times are significant because while data is being transferred into the operating system, the application must leave the buffers holding the data undisturbed, and until the data reaches the file, other applications cannot read it.

We can associate a rate with each of these times:

- *Application I/O rate* denoted by $Rate_{app}(n)$ which is equal to $n/Time_{app}(n)$.
- *Actual I/O rate* denoted by $Rate_{actual}(n)$ which is equal to $n/Time_{actual}(n)$.

3.1 Model with unlimited compute node memory and no I/O system memory

Assume the compute nodes have unlimited memory and the I/O system has no memory. In this case, the I/O node moves data directly between the compute nodes and the disk. Therefore $Time_{app}(n) = Time_{actual}(n)$ and this implies that $Rate_{app}(n) = Rate_{actual}(n)$. Let us call this $Rate(n)$.

As we increase n , we would expect to see $Rate(n)$ to start low at first due to overheads, and to quickly increase to some $Rate_{limit}$ corresponding to the raw I/O rate of the physical disk (see the first graph in Figure 2). After this point there is no change in I/O performance when the number of compute nodes or the size of the computation is scaled upwards. So if we scale the number of compute nodes by a factor of f , the time for I/O changes by the same factor f .

3.2 Model with a finite I/O buffer on the I/O node

Now assume that the I/O node has a disk buffer of finite size b bytes. In this model and the next we limit our discussion to output, for clarity. Parallel I/O in this model consists of two processes. Data is transferred from the application through the operating system to the disk buffer and from the disk buffer to the disk concurrently. Due to the relative bandwidths of current disks and interconnection networks, it makes sense to assume that the transfer of data from the application's buffer into the disk buffer happens much faster than the transfer from the disk buffer to the disk.

As the second graph in Figure 2 shows, when the total amount of output (n) that the application performs is less than the size of the disk buffer (b) then the application output rate will be much higher than the actual output rate, since the operating system can quickly copy the contents of the application's buffers

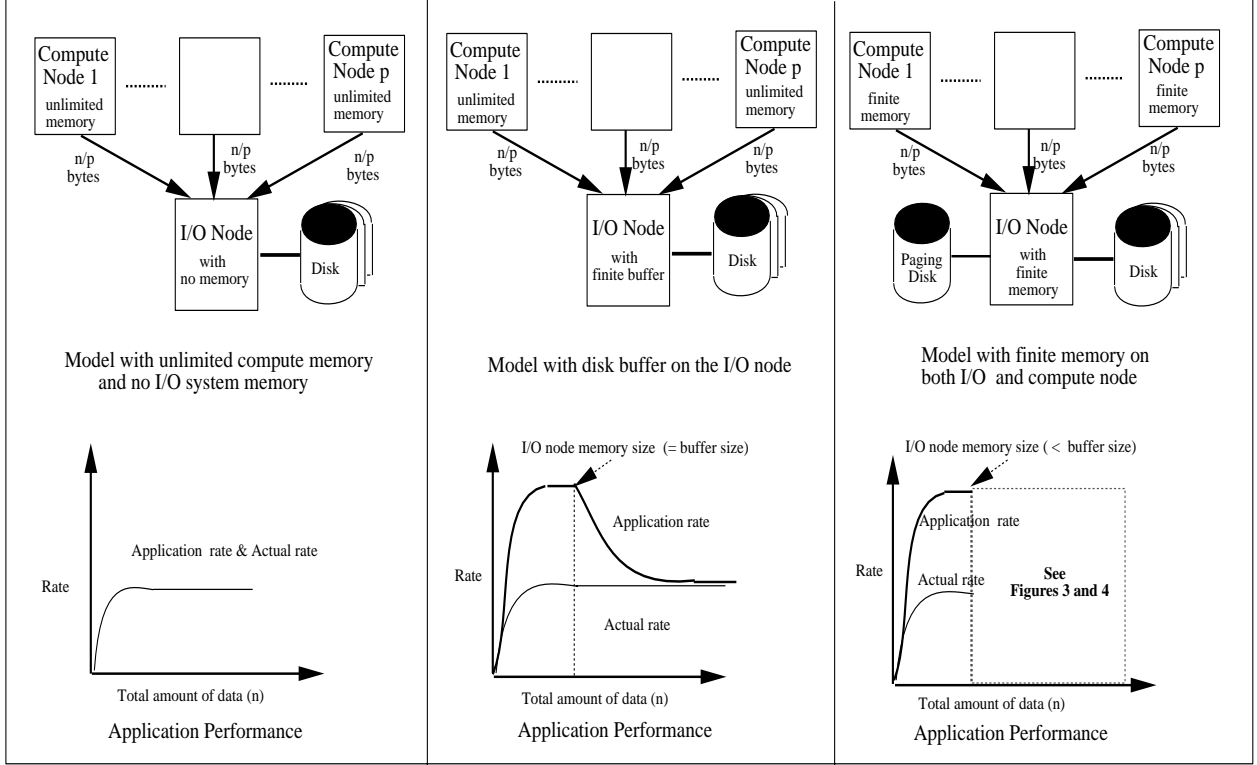


Figure 2: **ABSTRACT MODELS OF PARALLEL I/O.** The *Application I/O time* is the time the system call takes to move data between the application's buffer and the top level of the I/O system. *Actual I/O time* is the actual time for the I/O operation to complete *i.e.*, the time it takes for data to go all the way between the application and the disk. n is the total amount of data in the I/O operation. *Application I/O rate* is $n/Time_{app}(n)$ and *Actual I/O rate* is $n/Time_{actual}(n)$.

into the I/O node's buffer and then release the application's buffers. Therefore the system calls can return before the data is actually written to the disk. When n is very small, overheads will slow the application output rate, so a graph of the application output rate as a function of n will start low, rise quickly, then asymptotically approach a value corresponding to the peak rate at which the operating system can transfer data from the compute nodes into the I/O node's buffer.

However when the amount of output performed is greater than the disk buffer size, the system calls will start to take longer to return, for the following reason. The transfer rate from the operating system to the disk buffer will be at the same rate at which data is being transferred from the disk buffer to the disk. Therefore at b bytes of output, the graph of the application output rate will start dropping, asymptotically approaching the actual output rate.

More formally:

$$n < b \Rightarrow Rate_{app}(n) > Rate_{actual}(n)$$

$$n \gg b \Rightarrow Rate_{app}(n) = Rate_{actual}(n)$$

3.3 Model with virtual memory on compute and I/O nodes

Actual parallel systems have finite memory on both the compute and I/O nodes but provide virtual memory through paging mechanisms. In this section we extend our model to include virtual memory on both the compute and I/O nodes. Let the size of the disk buffer be b bytes and let the physical memory size on both the compute nodes and the I/O node be m bytes, where $b > m$ (so the entire buffer is too large to fit into the I/O node's memory).

For $n < m$ the behavior of this model is the same as that of the model in the previous section. When $n > m$ paging starts on the I/O node. The behavior of the model will be complex, and will be determined by the paging algorithm and the raw I/O rates of the paging disk and the actual disk. Also for $n > p * m$ paging will occur on both the I/O and compute nodes.

Analytical modeling of this behavior is difficult so we have developed a program to simulate this behavior (see Figures 3 and 4).

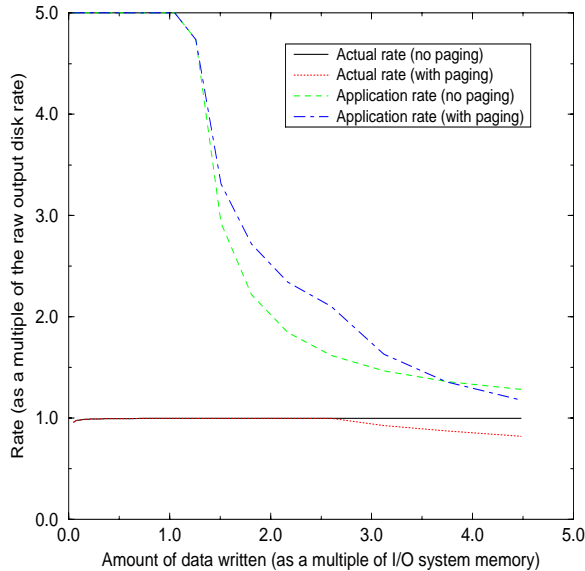


Figure 3: **COMPARING OUTPUT PERFORMANCE** with and without virtual memory on the I/O system (compares performance of models in sections 3.2 and 3.3). (This data was obtained from a simple virtual memory simulator). We assume the raw output disk bandwidth is one fifth the interconnection network bandwidth, and that the raw paging disk bandwidth is one half the raw output disk bandwidth. This preliminary result suggests that the use of virtual memory for the I/O system can improve application output rates at the expense of actual output rates.

Several further extensions to the model are possible. They include adding the effects of the interconnection network, the effect of the disk configuration *i.e.*, the mapping between I/O requests and the disk array, and the effect of multiple I/O nodes.

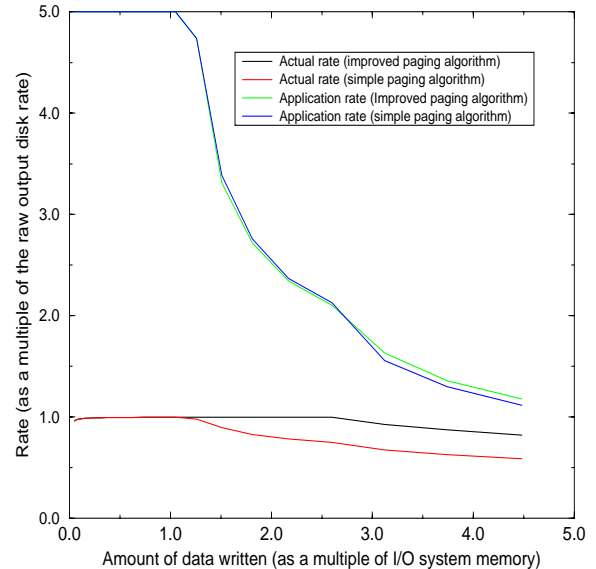


Figure 4: **COMPARING PAGING ALGORITHMS** for the model with virtual memory on the I/O nodes (simulator data). Both actual and application performance start to drop (due to paging) when the total amount of data written equals the total I/O system memory. The severity of the drop depends on the paging algorithm used. With a simple demand paging algorithm, the actual rate drops to 60% of the raw disk rate when 5 times the I/O system memory size is written. With an improved paging algorithm which incorporates early page-in of paged blocks to unused I/O system memory, the actual rate drops to 85% of the raw disk rate, a smaller drop than for the simple algorithm.

4 Experimental evaluation of parallel I/O performance on the Paragon and CM-5

On real machines, large reads and writes are often broken into series of smaller ones to improve I/O performance. Let us define a *parallel write operation* as a single parallel write system call performed on each compute node of a parallel machine, and a *parallel output operation* as a series of one or more parallel write operations with no intervening computation. As mentioned in the previous section, there are two signif-

icant amounts of time associated with a given parallel output operation: the *application output time* and the *actual output time*. Also there are two data transfers rates *application output rate* and *actual output rate* that are associated with these times. We define *parallel read operation*, *parallel input operation*, *application input rate*, and *actual input rate* similarly. Our experimental results for input are all in terms of actual input rate. The reason being that after initiating the read operation, there will be some period of time during which no data is being transferred into the user's program, while the data is being read from the disk and sent through the network. But none of the current parallel operating systems that we are aware of provide a way to measure that time. So we have no way of measuring the remaining time when data is being transferred into the application's buffers. Therefore, even though it makes sense to talk about an application input rate for a parallel read operation, we have no way of measuring that rate.

4.1 Intel Paragon: Experiments and Results

We ran our Intel Paragon experiments on the Paragon at Indiana University, which has 92 compute nodes, 2 I/O nodes and 4 partition managers. The file partition we used has 2 RAID-3 arrays; each array has 5 disks for a total of 10 disks. We were the only users on the machine during most of the experiments.

The I/O nodes on this machine are in a service partition. This is not the most optimal configuration since a heavyweight OSF/1 kernel that runs all UNIX commands issued by the users resides on the service nodes, using up potential I/O buffer space.

On the Paragon we started with a simple test program that linearly increased the total amount of I/O, and measured the I/O rates for the `M_SYNC` mode. We successively added the following optimizations that the Paragon User's Guide[2] suggests:

1. Align read/write buffers to page boundaries¹.
2. Use the paragon's record (`M_RECORD`) parallel I/O mode. This mode requires that the same amount of data be read or written by each processor in any given parallel I/O operation.
3. Break large write operations into multiple smaller operations, where the amount of data written by each processor is either equal to the stripe unit

¹If properly aligned buffers are not used, the software must copy the data to new, aligned buffers, worsening performance.

size, or is an integer multiple of the full stripe size. This makes optimum use of file striping, keeping all the I/O nodes busy at once.

4. Begin each read and write request on a file system block boundary.

The results presented in the paper are for the final most optimized test programs.

4.1.1 Parallel Input Performance

Figure 5 is a graph of the empirically measured actual input rate for varying sizes of parallel input operations, each comprised of a single parallel read operation, for 2, 4, and 8 processors.

The Paragon User's Guide indicates that parallel write performance can be improved by breaking large parallel write operations into a series of smaller operations, where the amount of data written by each operation is equal to the stripe unit size. We found that breaking up large parallel read operations into smaller read operations does not improve performance.

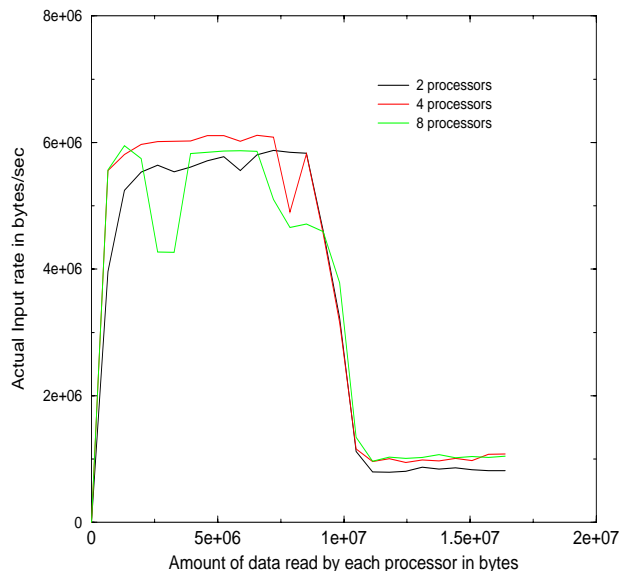


Figure 5: **INPUT PERFORMANCE** of the Paragon in terms of the actual input rate. Notice the performance drop when the input size exceeds the physical memory of the compute nodes, causing paging

4.1.2 Parallel Output Performance

The output rates are graphed as functions of the total amount of data output in a parallel output operation. Each parallel output operation was comprised of a series of smaller parallel write operations of size equal to the I/O system's stripe unit size.

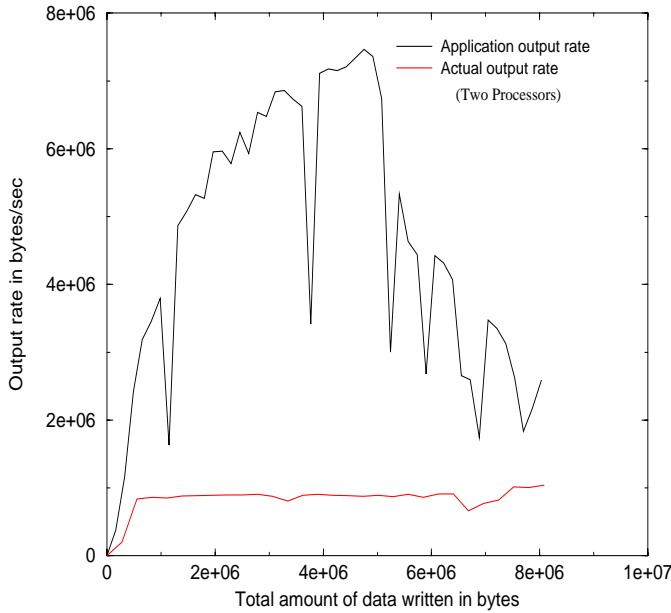


Figure 6: **COMPARING APPLICATION AND ACTUAL OUTPUT PERFORMANCE** measured using two compute nodes on the paragon. The graph shows that the actual output rate remains nearly constant, no matter how much data is written. The graph of the application output rate, on the other hand, rises to a peak around 5MB then suddenly begins to drop. A possible cause of this behavior becomes apparent in Figure 7.

In Figure 6 we see a sudden drop in the application output rate beyond a certain point. So, what causes the sudden drop in the application output rates of the parallel write operations? Our model predicts that if small parallel write operations are repeatedly performed, then the application output rates of those operations should be high until the memory of the I/O nodes is filled, at which point the application output rates should suddenly drop due to the extra time required for paging on the I/O nodes. Indeed this is what we observe. If this explanation is correct, then

the location of the peak in the application output rate of the parallel output operations should depend only on the total amount of data written, and not on the number of compute nodes doing the writing. This is what we observed experimentally. Figure 7 shows a comparison of application output rates of parallel output operations of varying size for 2, 4, and 8 processors. The three graphs all have the same shape, and the peak in all the three graphs appears at the same place.

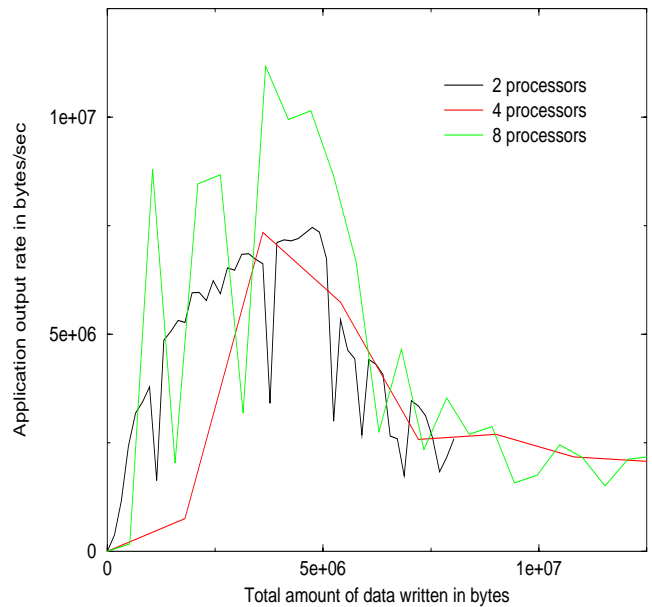


Figure 7: **COMPARING APPLICATION OUTPUT RATES** for varying numbers of processors measured on the paragon. Notice that the performance drop happens at around the same point (close to 5MB) for all the graphs. We believe this drop is caused by the I/O system memory filling up.

Our model predicts that if after writing enough data to fill the I/O nodes' memory, the compute nodes wait for that data to be completely written to the disk (freeing the I/O nodes' memory once again), then the compute nodes should see a high application output rate when the writing is resumed. The above behavior is similar to what we observed in our experiments as shown in Figure 8. The line corresponding to the case where no waiting was done was obtained as follows: a series of small parallel write operations were per-

formed to a single file, and the application output rate was recorded for each write. These rates were plotted in the graph from left to right, the x coordinate of the rate of the i 'th write being determined by the sum of the sizes of all the previous writes in the series. So the rate at x coordinate i is the application output rate that will be observed from a small write performed immediately after i bytes of data have been written. The line corresponding to waiting was obtained similarly, except that at the point when enough data had been written to fill the I/O nodes' memory, at 3 Mbytes of data written per processor, the `fsync` system call was used to wait for that data to be completely written to the disk, then writing was resumed.

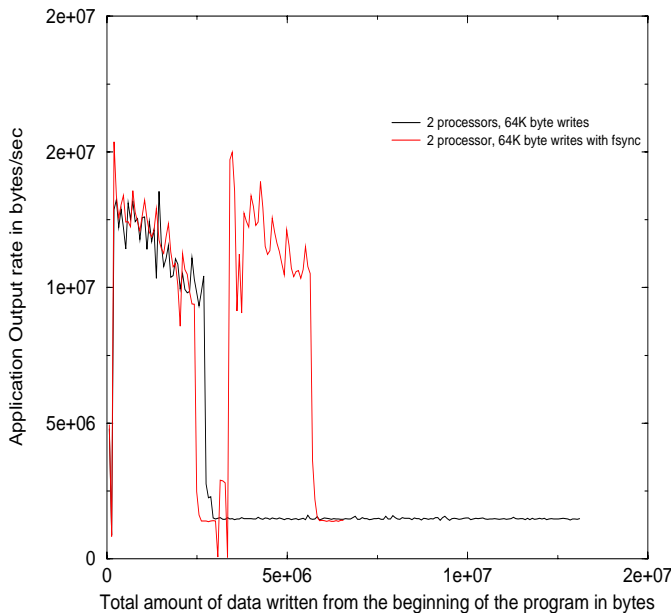


Figure 8: **TWO TIME VIEW GRAPHS OF APPLICATION OUTPUT RATES** of small repeated parallel write operations. The rate starts high in both graphs, then suddenly drops when the I/O system's memory is filled, due to the onset of paging. Then in one graph, an `fsync()` system call is performed (at about 3 MB), forcing the I/O system's memory to clear before writing is resumed, at which point a second hump appears in the graph. In the other graph, no `fsync()` is performed, and the rate remains low after the initial drop.

Figure 9 examines the implications of paging on the

I/O nodes when we write large amounts of data.

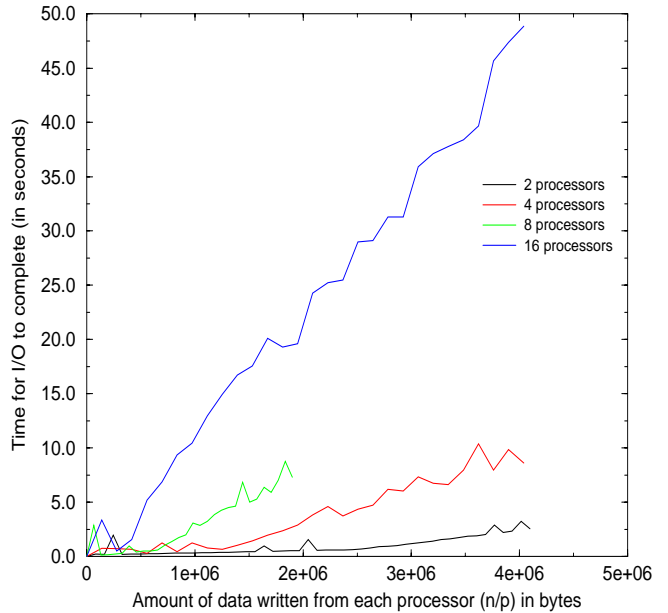


Figure 9: **WHEN THE AMOUNT** of data written exceeds the I/O system's memory, doubling the amount of data written causes the application output time to increase by a factor of 4. The following is a possible explanation. If the I/O system's memory fills up paging will begin, causing disk I/O time to be added to the application output time, which before was just the time to transfer data through the interconnection network to the I/O system's memory. Consequently, when the I/O system is saturated, paging disk I/O time will begin to dominate the application output time. This could account for the dramatic slowdown.

4.2 Thinking Machines CM-5

We ran these experiments on the machines at NCSA and at UMIACS, Maryland. We present results only for the CM-5 at UMIACS. The CM-5 at UMIACS has 32 compute nodes, 2 I/O nodes, 1 partition manager. Our results are preliminary since we used the wall clock timer for our measurements with other users on the system, since the virtual timers on the CM-5 cannot be used for I/O measurements.

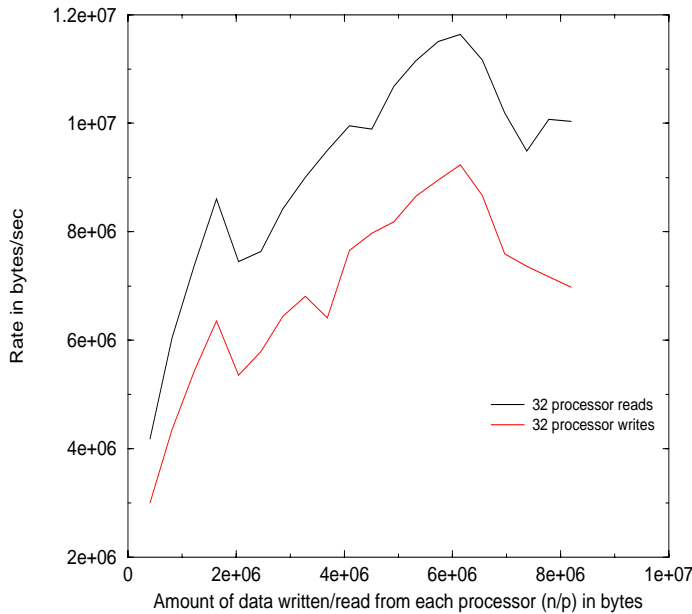


Figure 10: **INPUT AND OUTPUT PERFORMANCE** of application output and actual input rates on the Maryland CM-5. Notice the performance drop for both input and output rates beyond 6MB

5 Related work

Kwan and Reed [6] study the performance of the Scalable File System on the NCSA CM-5. Their motivation was to compare the performance of the CM-5 parallel I/O interfaces of CM-Fortran and C/CMMD. Their C/CMMD experiments are similar to ours. Our results on the CM-5 are preliminary but are comparable with theirs.

Bordawekar, Rosario and Choudhary [7] present an experimental performance evaluation of the Touchstone Delta Concurrent File System. The actual read (6 MB/sec) and actual write (1.2 MB/sec) rates we measured on the IU Paragon are a lot lower than the rates they report for the Intel Touchstone Delta (also reported in [6]).

6 Conclusions

We identify two important data transfer rates for I/O: the application rate and the actual rate. The ap-

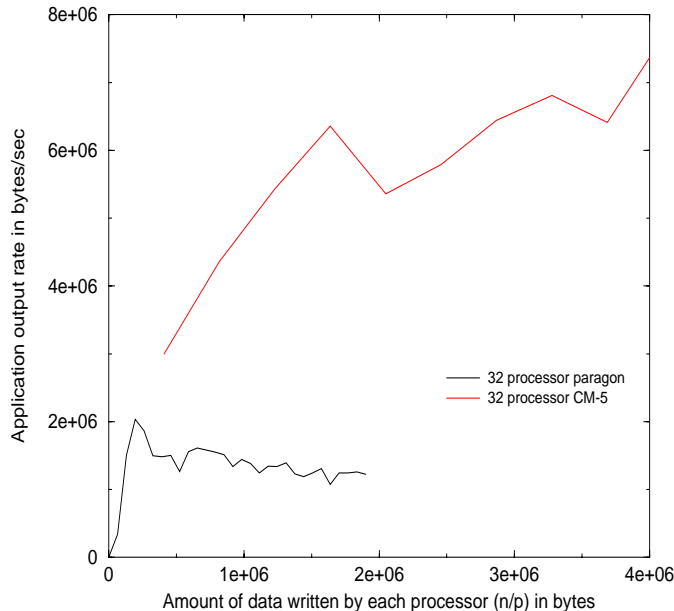


Figure 11: **COMPARING THE APPLICATION OUTPUT PERFORMANCE** of the IU's Intel Paragon and the Maryland's CM-5. Both measurements are for 32 processors. Each Paragon output operation was comprised of a series of parallel write operations each of a size equal to the stripe unit size. Also each CM-5 input/output operation was comprised of a single parallel read/write operation.

plication rate is the rate at which data is transferred between the user's buffer and the operating system; the actual rate is the rate at which the data is transferred between the user's buffer and the disk. Both rates are useful for evaluating I/O performance.

We develop three successively more detailed models of parallel I/O systems and describe the I/O performance of a simple I/O intensive application for each model. Many further extensions to the most detailed model are possible (see section 3.3). These models are useful for explaining parallel I/O performance to developers of applications requiring high performance I/O. We examine the effect of I/O system virtual memory on I/O performance through simulation. We then describe the I/O performance of actual parallel I/O systems on the Intel Paragon and the Thinking Machines CM-5.

We identify the following optimizations for improv-

ing I/O performance:

- The user's buffer should be properly aligned in memory so that the operating system doesn't have to make an aligned copy of the buffer.
- Striping mechanisms provided in parallel I/O systems should be used effectively to balance the I/O load among the components of the parallel I/O system.
- If a fast application rate is desired (for example, in applications that can overlap output with computation involving the user's output buffer memory), then care should be taken to write no more data than will fit into the I/O system's memory in a given output cycle.
- Our preliminary simulation results suggest that the use of virtual memory for the I/O system can improve application output rates at the expense of actual output rates. However, more work is needed to determine how varying the relative bandwidths of the paging disk and the output disk affects this result.
- Our simulation data indicate that if virtual memory is to be used for parallel I/O systems, specialized paging algorithms, such as early page-in to unused I/O system memory, are necessary to maintain good actual output performance.
- If virtual memory is used for the I/O system then the user can improve actual output performance if I/O system paging is avoided by writing only enough data to fill the I/O system memory, then waiting until that data is completely written to the disk (using the `fsync()` system call) before writing more data.

Several efforts are under way to provide parallel I/O libraries for the current generation of parallel languages. The above optimizations should be included in those libraries, making parallel I/O simpler and more efficient for application developers.

Acknowledgements

We would like to thank Dennis Gannon for his encouragement and direction during this work. Thanks to NCSA and UMIACS for time on their CM-5's. Thanks to Thomas Kwan for his comments about the CMMD timers on the CM-5. Thanks also to Peter Molnar at University Computing Services for technical

help with the Intel Paragon and for interacting with the Intel SSD division to answer our questions. This research is supported in part by ARPA under contract AF 30602-92-C-0135, the National Science Foundation Office of Advanced Scientific Computing under grant ASC-9111616.

References

- [1] Thinking Machines Corporation. *CMMD Reference Manual*. Chapter 12.
- [2] Intel Supercomputing System Division. *Paragon Users Guide*. Chapter 5 and Chapter 8.
- [3] Peter Corbett and Dror Feitelson. Design and implementation of the vesta parallel file system. In *Proceedings of Scalable High Performance Computing Conference, SHPCC94*, May 1994.
- [4] Juan Rosario and Alok Choudhary. High-performance I/O for massively parallel computers: Problems and Prospects. *IEEE Computer*, pages 59–68, March 1994.
- [5] Mark Henderson, Bill Nickless, and Rick Stevens. A scalable high-performance I/O system. In *Proceedings of Scalable High Performance Computing Conference, SHPCC94*, May 1994.
- [6] Thomas Kwan and Daniel Reed. Performance of the CM-5 scalable file system. In *Proceedings of International Conference on Supercomputing 94*, July 1994. Also accessible on the World Wide Web at <http://bugle.cs.uiuc.edu/>.
- [7] R Bordawekar, J.M. Del Rosario, and A Choudhary. Experimental performance evaluation of touchstone delta concurrent file system. In *Proceedings of International Conference on Supercomputing 93*, July 1993.
- [8] James C. French, Terrence W. Pratt, and Mriganka Das. Performance measurement of the concurrent file system of the intel iPSC/2 hypercube. *Journal of Parallel and Distributed Computing*, 17:115–123, 1993.
- [9] Alok Choudhary. Parallel I/O systems: Guest editor's introduction. *Journal of Parallel and Distributed Computing*, 17:1–3, 1993.
- [10] The parallel i/o archive. Available on the World Wide Web at <http://www.cs.dartmouth.edu/pario.html>.