An Experimental Evaluation of the Parallel I/O Systems of the IBM SP and Intel Paragon Using a Production Application*

Rajeev Thakur *William Gropp* *Ewing Lusk*

Mathematics and Computer Science Division
Argonne National Laboratory
9700 S. Cass Avenue
Argonne, IL 60439

{thakur, gropp, lusk}@mcs.anl.gov

Abstract

This paper presents the results of an experimental evaluation of the parallel I/O systems of the IBM SP and Intel Paragon. For the evaluation, we used a full, three-dimensional application code that is in production use for studying the nonlinear evolution of Jeans instability in self-gravitating gaseous clouds. The application performs I/O by using library routines that we developed and optimized separately for parallel I/O on the SP and Paragon. The I/O routines perform two-phase I/O and use the PIOFS file system on the SP and PFS on the Paragon. We studied the I/O performance for two different sizes of the application. We found that for the small case, I/O was faster on the SP, whereas for the large case, I/O took almost the same time on both systems. Communication required for I/O was faster on the Paragon in both cases. The highest read bandwidth obtained was 48 Mbytes/sec. and the highest write bandwidth obtained was 31.6 Mbytes/sec., both on the SP.

Key words: parallel I/O, I/O-intensive applications, I/O characterization, performance evaluation

*This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Computational and Technology Research, U.S. Department of Energy, under Contract W-31-109-Eng-38; and by the Scalable I/O Initiative, a multiagency project funded by the Advanced Research Projects Agency (contract number DABT63-94-C-0049), the Department of Energy, the National Aeronautics and Space Administration, and the National Science Foundation.

DISCLAIMER

1 Introduction

It is widely recognized that, in addition to fast computation and communication, parallel machines must also be able to provide fast parallel I/O. There is no consensus, however, as to the best architecture for parallel I/O or the best kind of parallel file system. Several alternatives have been proposed, some of which are being used in current-generation parallel machines. The exact parallel I/O hardware and software for a teraflops machine remains a research issue.

This paper presents the results of an experimental evaluation of the parallel I/O systems of two different, state-of-the-art parallel machines—the IBM SP at Argonne National Laboratory and the Intel Paragon at Caltech—using a real, I/O-intensive parallel application. These machines are the two testbeds for the Scalable I/O Initiative[1]. We chose a full, three-dimensional application code that is in production use for studying the nonlinear evolution of Jeans instability in self-gravitating gaseous clouds. We wrote special parallel I/O routines for this application and optimized them separately for the SP and Paragon. We instrumented these I/O routines, ran the application on both systems, and analyzed the resulting trace files. Two sizes of the application were considered. We found that in the small case, I/O was faster on the SP, whereas in the large case, I/O took almost the same time on both systems. Communication required for I/O was faster on the Paragon in both cases.

The rest of this paper is organized as follows. Section 2 gives a brief overview of related work in the area of I/O characterization of parallel applications and performance evaluation of parallel file systems. In Section 3, we discuss the application and its I/O requirements. We describe the experimental setup in Section 4. The I/O characteristics on the SP and Paragon are presented in Sections 5 and 6, respectively. We draw overall conclusions in Section 7.

2 Related Work

Numerous studies have focused on I/O characterization of applications, and performance evaluation of file systems, on sequential and vector machines. Only recently, however, have similar studies been conducted on parallel machines. Because of space limitations, we discuss here only the previous work done on parallel machines.

We first consider I/O characterization studies. Kotz and Nieuwejaar [16] presented the results of a three-week tracing study in which all file-related activity on the Intel iPSC/860 at NASA Ames Research Center was recorded. They did not instrument individual applications; instead, they instrumented the Concurrent File System (CFS) library routines called by all applications, and studied the file-related activity of all applications together. A similar study on the CM-5 at

[1] Information about the Scalable I/O Initiative is available at `http://www.cacr.caltech.edu/SIO/`

the National Center for Supercomputing Applications was performed by Purakayastha et al. [21]. The results of these two studies were compared and contrasted in [19]. The results showed that file sizes were large, request sizes were fairly small, data was accessed in sequence but with strides, and I/O was dominated by writes.

Crandall et al. [7] analyzed the I/O characteristics of three parallel applications on the Intel Paragon at Caltech. They found a wide variety of access patterns, including both read-intensive and write-intensive phases, large as well as small request sizes, and both sequential and irregular access patterns. Baylor and Wu [3] performed a study of the I/O characteristics of four parallel applications on an IBM SP using the Vesta parallel file system. They found I/O request rates on the order of hundreds of requests per second, mainly small request sizes, and strong temporal and spatial locality. Acharya et al. [1] reported their experience in tuning the performance of four applications on an IBM SP. Reddy and Banerjee [22] studied the I/O activity of five applications from the Perfect benchmark suite on an Alliant multiprocessor and found only sequential accesses. Del Rosario and Choudhary [10] provided an informal summary of the I/O requirements of several Grand Challenge applications. Cypher et al. [8] discussed the I/O requirements of eight parallel scientific applications.

Researchers have also evaluated the performance of parallel file systems. Bordawekar et al. [4] performed a detailed performance evaluation of the Concurrent File System (CFS) on the Intel Touchstone Delta. Kwan and Reed [17] measured the performance of the CM-5 Scalable File System. Feitelson et al. [11] studied the performance of the Vesta file system running on an SP-1. Nieuwejaar and Kotz [18] presented performance results for the Galley parallel file system, a new parallel file system developed at Dartmouth College. Several studies have also been made of the performance of the Concurrent File System (CFS) on the Intel iPSC/2 and iPSC/860 hypercubes [5, 12, 20].

In an earlier work, we studied the I/O characteristics of a different application on the SP and Paragon [24]. For that study, we used a 2D astrophysics application that performed I/O using the sequential Unitree file system on the SP and the PFS file system on the Paragon. For this paper, we used a completely different 3D application that is much more data intensive and performs actual parallel I/O (two-phase I/O) using the parallel file system PIOFS on the SP and PFS on the Paragon.

3 The Application and its I/O

The application we used is a parallel production code developed at the University of Chicago to study the nonlinear evolution of Jeans instability in self-gravitating gaseous clouds, a process

considered to be the basic mechanism for the formation of stars and galaxies. The code solves the equations of compressible hydrodynamics with the inclusion of self-gravity. The computational algorithm consists of the piecewise parabolic method [6] to solve the compressible Euler equations and a multigrid elliptic solver to compute the gravitational potential.

The code uses the Chameleon library [14] for communication. The Chameleon communication routines provide a portable interface to the native communication library on a given machine. The application originally also used the Chameleon library for I/O [13], but we found that the Chameleon I/O routines were not well optimized for parallel I/O on the SP and Paragon. Therefore, we wrote special routines with the same interface as the Chameleon I/O library routines, but separately optimized for the SP and Paragon (see Section 4 for further details about the I/O routines). The application performs all I/O via calls to these optimized routines. Therefore, the application code is directly portable across the two machines. The application is written in Fortran whereas the I/O routines are in C.

The application uses several three-dimensional arrays. A typical array size is $256 \times 256 \times 128$. The arrays are distributed in a (block,block,block) fashion among the processors. All data fits in main memory. The program is iterative, and every few iterations, several arrays are written to files for three purposes—data analysis, checkpointing (restart), and visualization. The ordering of data in all files is required to be the same as it would be if the program were run on a single processor.

The data analysis file begins with six variables (real numbers) that have the same value across all processors, followed by six arrays appended one after another. The arrays are stored in column-major (Fortran) order. A new data analysis file is written during each dump. The restart file has the same structure as the data analysis file, but the same file is overwritten during each dump. Although computation is performed in double precision, single-precision data is written to the data analysis and restart files. The visualization data consists of four separate files. Each file begins with six variables (real numbers) that have the same value across all processors, followed by one array of character data. New visualization files are written during each dump.

In other words, the number of files created by the application is one per dump for data analysis, one for restart, and four per dump for visualization. For a $256 \times 256 \times 128$ grid, the size of each of the data analysis files and the restart file is approximately 192 Mbytes, and the size of each of the visualization files is approximately 8 Mbytes.

4 Details of the Experiments

We describe the SP and Paragon systems, the I/O routines we used, and the various experiments that were performed.

4.1 Machine Specifications

We used the IBM SP at Argonne National Laboratory and the Intel Paragon at Caltech, which are the two testbeds for the Scalable I/O Initiative.

4.1.1 IBM SP

The IBM SP at Argonne National Laboratory has 128 nodes. Eight of these nodes are RS/6000 Model 970 with 256 Mbytes of main memory; the other 120 nodes are RS/6000 Model 370 with 128 Mbytes of main memory. The nodes are interconnected by a high-performance omega switch. The operating system on each node is AIX 3.2.5.

The parallel file system on the SP is IBM PIOFS. The eight Model 970 nodes function as servers for PIOFS. Each server has 3 Gbytes of local SCSI disks, resulting in a total PIOFS storage capacity of 24 Gbytes. At the time we performed our experiments, the system was configured such that the eight Model 970 nodes functioned exclusively as I/O servers; users were not allowed to run any compute jobs on them. In other words, the system effectively had 120 compute nodes and 8 I/O server nodes.

In PIOFS, a file is distributed across multiple I/O server nodes. The file is logically organized as a collection of *cells*, where a cell is a piece of the file stored on a particular server node. A file is divided into a number of *basic striping units* (BSUs), which are assigned to cells in a round-robin manner. Cells in turn are assigned to server nodes in a round-robin manner. The default number of cells is equal to the number of server nodes, and the default BSU size is 32 Kbytes.

PIOFS also supports logical partitioning of files. Processors can specify a logical *view* of the data to be accessed and then read/write the data in a single operation, even though the data may not be located contiguously in the file. However, it is currently not clear to us how to specify the view for a three-dimensional array distributed in all three dimensions, as in our application. (We do understand how to do so for a three-dimensional array distributed in only two dimensions.) Therefore, in this paper, we do not use the logical partitioning feature of PIOFS. We are working with IBM on this problem, and if we find a solution, we will use it and report the results in the final version of this paper. At present, we perform all seeks, reads, and writes explicitly.

4.1.2 Intel Paragon

The Intel Paragon at Caltech has 512 compute nodes, each of which is an Intel i860/XP microprocessor. Each compute node has 32 Mbytes of main memory. The nodes are connected by a two-dimensional mesh interconnection network. The operating system on the machine is Paragon/OSF R1.3.3. The parallel I/O system on the Paragon consists of 21 dedicated I/O nodes that do not

run any compute jobs. Each I/O node has 32 Mbytes of main memory and is connected to a 4.8 Gbyte RAID-3 disk array. Intel's Parallel File System (PFS) provides parallel access to files.

A PFS file system consists of one or more *stripe directories*. Each stripe directory is usually the mount point of a separate Unix file system (UFS). Just as a RAID subsystem collects several disks into a unit that behaves like a single large disk, a PFS file system collects several file systems into a unit that behaves like a single large file system. PFS files are divided into smaller *stripe units* and distributed in a round-robin fashion across the stripe directories that make up the PFS file system. The Paragon at Caltech has 16 stripe directories, and the default stripe unit is 64 Kbytes.

4.2 Parallel I/O Routines

In this application, three-dimensional arrays are distributed among processors in a (block,block,block) manner. Each distributed array must be written to a common file such that the data in the file corresponds to the global array in column-major (Fortran) order. The original Chameleon I/O routines perform this task by having all processors send their data to processor 0; only processor 0 actually writes data to the file. This approach was clearly inefficient because of the sequential nature of the I/O and also the communication bottleneck caused by the all-to-one communication pattern.

To overcome these limitations of the Chameleon I/O routines, we wrote new routines, having the same interface as Chameleon I/O, but performing actual parallel I/O on the SP and Paragon. These routines use PIOFS on the SP and PFS on the Paragon. We optimized the routines separately for the two systems; for example, on the Paragon, we used the gopen() call for faster opens and the M_ASYNC mode for faster reads and writes. The application program interface (API) did not change; therefore, we did not have to change the application code.

Since the local array of each processor in this application is not located contiguously in the file, an attempt by any processor to read/write its local data directly would result in too many small read/write requests. We eliminated this problem by using two-phase I/O [9], a technique for reading/writing distributed arrays efficiently. In two-phase I/O, as the name suggests, a distributed array is read or written in two phases. For writing a distributed array, in the first phase, the array is redistributed among the processors such that in the new distribution, each processor's local data is located contiguously in the file. For our application, this required a redistribution from (block,block,block) to (*,*,block); in other words, after the first phase, only the third dimension of the array was distributed. In the second phase of two-phase I/O, processors write their local data at appropriate locations in the file, in parallel and with a single write operation. This two-phase method eliminates the need for several small reads/writes and also has a fairly balanced all-to-many communication pattern. Reading a distributed array using two-phase I/O involves

each processor reading a contiguous block in the first phase and then redistributing it in the second phase. (Note that if we were able to use the logical partitioning feature of PIOFS, we need not have used two-phase I/O on the SP. Nevertheless, we are interested in comparing the performance of logical partitioning and two-phase I/O.)

The application also requires six variables (real numbers) having the same value across all processors to be written to a file. To do so, we collected the six variables into a single buffer on processor 0 only and had processor 0 write the buffer to the file. For reading during a restart, processor 0 read the buffer and broadcast it to other processors.

4.3 Instrumentation and Analysis

To study the I/O behavior of the code, we instrumented the I/O routines using the Pablo performance analysis environment [2, 23]. We instrumented all respective calls to open, close, read, write, and seek, and also all communication required for I/O. We ran the instrumented code on both the SP and Paragon and collected trace files. The traces were visualized and analyzed by using Upshot [15], a tool for studying parallel program behavior.

4.4 Experiments

On both machines, we ran two sizes of the application:

1. **Small:** For this case, we used a $128 \times 128 \times 64$ mesh on 8 processors. We restarted the code from a previously created restart file and ran it for 20 iterations. The dumps for data analysis, restart, and visualization were performed every 5 iterations.

2. **Large:** For this case, we used a $256 \times 256 \times 128$ mesh on 64 processors. Since the application runs only on a power-of-two number of processors and the SP at Argonne currently has 120 compute nodes, 64 was the maximum number of processors we could use. For a fair comparison, we used the same number on the Paragon. We restarted the code from a previously created restart file and ran it for 5 iterations, with all the dumps performed after the fifth iteration.

For each size, we ran the code several times. The results reported are for the case that took the lowest time. The application only performs writes except when restarting from a checkpoint. To be able to measure the read performance as well, we restarted the code from a checkpoint each time. Since the application is iterative, a complete run to convergence could take more than 10,000 iterations. To keep the trace files manageable, we ran the code only for a few iterations, as specified above. The I/O behavior in the remaining iterations is assumed to be similar to that in the first few iterations.

5 Results on the SP

Table 1 shows the file sizes and number of files of each type for the small case. The data analysis and restart files were 24 Mbytes each, and the visualization files were 1 Mbyte each. The same restart file was overwritten in each dump, whereas new data analysis and visualization files were written each time.

Table 2 shows the total number of I/O operations of each type and the total time taken for each type of I/O operation. The average time per processor was calculated as the total time across all processors divided by the number of processors. There were 200 open calls in all, and the open calls were fairly expensive, taking more time than for read, close, or seek. Close and seek took almost negligible time. The write operations required a large amount of time, which was expected since the application is write-intensive. Communication for I/O took even more time than the write operations. This was not the case on the Paragon, however, as we discuss in Section 6.

Table 1: File sizes for the small case—128 × 128 × 64 mesh on 8 processors

Type	No. of Files	Size (MB)
Data Analysis	1 per dump	≈ 24
Restart	1	≈ 24
Visualization	4 per dump	≈ 1

Table 2: I/O operations on the SP for the small case—128 × 128 × 64 mesh on 8 processors, 20 iterations

Operation	Total Count (all procs.)	Total Time (sec.) (all procs.)	Average Time (sec.) (per proc.)
Open	200	31.35	3.919
Close	200	1.693	0.217
Read	49	4.001	0.500
Write	536	52.67	6.584
Seek	1225	0.052	0.006
Communication (for I/O)	5608	83.39	10.42

Table 3 shows the distribution of the individual write sizes. Most of the reads and writes were large, since we used two-phase I/O. There were a few small reads and writes because the application requires reading/writing six variables (real numbers) at the beginning of the file. As explained in Section 4.2, these were read/written in a single operation by processor 0 only. The aggregate read

bandwidth across all processors, computed as the total data read divided by the average read time per processor, was 48 Mbytes/sec. The aggregate write bandwidth, computed similarly, was 31.6 Mbytes/sec.

Tables 4, 5, and 6 show the results for the large case on the SP. The data analysis, restart, and visualization files were eight times larger than the files created in the small case. In the large case, the seek operations took negligible time, close operations took a small amount of time, and read and open operations took a fairly large amount of time. The most expensive operations were communication for I/O and write. The sizes of the individual read and write operations were the same as in the small case because, although the mesh size was eight times larger, the number of processors was also eight times larger. The aggregate read bandwidth was 33.9 Mbytes/sec. and the aggregate write bandwidth was 17.4 Mbytes/sec., both of which were lower than in the small case.

Table 3: Details of read and write operations on the SP for the small case—128 × 128 × 64 mesh on 8 processors, 20 iterations

	Size Distribution			Total Data	Aggregate
Operation	24 B	128 KB	512 KB	Transferred (MB)	BW (MB/sec.)
Read	1	0	48	≈ 24	48.00
Write	24	128	384	≈ 208	31.59

Table 4: File sizes for the large case—256 × 256 × 128 mesh on 64 processors

Type	No. of Files	Size (Mbytes)
Data Analysis	1 per dump	≈ 192
Restart	1	≈ 192
Visualization	4 per dump	≈ 8

6 Results on the Paragon

Tables 7 and 8 show the results for the small case on the Paragon. The file sizes were the same as on the SP (Table 1). The counts and size distribution of the I/O operations were also the same as on the SP. However, the time taken by the I/O operations on the two machines was different, and the comparison shown in Table 9 is interesting. All I/O operations were more expensive on the Paragon. The open operations were 2.9 times slower on the Paragon, and the read and

Table 5: I/O operations on the SP for the large case—256 × 256 × 128 mesh on 64 processors, 5 iterations

Operation	Total Count (all procs.)	Total Time (sec.) (all procs.)	Average Time (sec.) (per proc.)
Open	448	358.5	5.601
Close	448	26.96	0.421
Read	385	362.5	5.664
Write	1030	1533	23.95
Seek	3235	0.138	0.002
Communication (for I/O)	109888	2142	33.47

Table 6: Details of read and write operations on the SP for the large case—256 × 256 × 128 mesh on 64 processors, 5 iterations

Operation	Size Distribution			Total Data Transferred (MB)	Aggregate BW (MB/sec.)
	24 B	128 KB	512 KB		
Read	1	0	384	≈ 192	33.90
Write	6	256	768	≈ 416	17.37

Table 7: I/O operations on the Paragon for the small case—128 × 128 × 64 mesh on 8 processors, 20 iterations

Operation	Total Count (all procs.)	Total Time (sec.) (all procs.)	Average Time (sec.) (per proc.)
Open	200	90.37	11.29
Close	200	7.586	0.948
Read	49	6.712	0.839
Write	536	97.16	12.15
Seek	1225	6.938	0.867
Communication (for I/O)	5608	70.17	8.771

Table 8: Details of read and write operations on the Paragon for the small case—128 × 128 × 64 mesh on 8 processors, 20 iterations

Operation	Size Distribution			Total Data Transferred (MB)	Aggregate BW (MB/sec.)
	24 B	128 KB	512 KB		
Read	1	0	48	≈ 24	28.61
Write	24	128	384	≈ 208	17.12

Table 9: Comparison of I/O performance on the SP and Paragon for the small case

Operation	Average Time per proc. on SP (sec.)	Average Time per proc. on Paragon (sec.)	Aggregate BW on SP (MB/sec.)	Aggregate BW on Paragon (MB/sec.)
Open	3.919	11.29		
Close	0.217	0.948		
Read	0.500	0.839	48.00	28.61
Write	6.584	12.15	31.59	17.12
Seek	0.006	0.867		
Communication (for I/O)	10.42	8.771		

write bandwidths were 60% and 55% of the bandwidths on the SP, respectively. However, the communication required for I/O was faster on the Paragon.

Tables 10 and 11 show the results for the large case on the Paragon. The file sizes, and the counts and size distribution of individual I/O operations were the same as on the SP. Table 12 shows a comparison of the I/O performance on the SP and Paragon for the large case. The difference in I/O performance of the two machines was not as much as in the small case. Most of the I/O operations took comparable time on the two machines, and writes were in fact faster on the Paragon. The most dramatic differences were that seeks were considerably slower on the Paragon, while communication for I/O was faster on the Paragon by a factor of two. An interesting observation is that the read and write bandwidths obtained on the SP were higher in the small case, whereas on the Paragon they were higher in the large case.

7 Conclusions

We have presented the results of an experimental evaluation of the parallel I/O systems of the IBM SP and Intel Paragon using a parallel production application. We used two different problem sizes

Table 10: I/O operations on the Paragon for the large case—$256 \times 256 \times 128$ mesh on 64 processors, 5 iterations

Operation	Total Count (all procs.)	Total Time (sec.) (all procs.)	Average Time (sec.) (per proc.)
Open	448	402.5	6.289
Close	448	36.68	0.573
Read	385	365.7	5.714
Write	1030	1429	22.33
Seek	3235	68.97	1.078
Communication (for I/O)	109888	1020	15.94

Table 11: Details of read and write operations on the Paragon for the large case—$256 \times 256 \times 128$ mesh on 64 processors, 5 iterations

Operation	Size Distribution			Total Data Transferred (MB)	Aggregate BW (MB/sec.)
	24 B	128 KB	512 KB		
Read	1	0	384	≈ 192	33.60
Write	6	256	768	≈ 416	18.63

Table 12: Comparison of I/O performance on the SP and Paragon for the large case

Operation	Average Time per proc. on SP (sec.)	Average Time per proc. on Paragon (sec.)	Aggregate BW on SP (MB/sec.)	Aggregate BW on Paragon (MB/sec.)
Open	5.601	6.289		
Close	0.421	0.573		
Read	5.664	5.714	33.90	33.60
Write	23.95	22.33	17.37	18.63
Seek	0.002	1.078		
Communication (for I/O)	33.47	15.94		

for this study. We found that in the small case, all I/O operations (open, close, read, write, seek) were faster on the SP, though the communication required for I/O was faster on the Paragon. In the large case, I/O operations took almost the same time on both machines, though communication was much faster on the Paragon. On the Paragon, the I/O performance was better in the large case than in the small case, whereas on the SP, the I/O performance was better in the small case.

We note that these performance results are for current versions of the operating systems on the two machines—AIX 3.2.5 on the SP and OSF R1.3.3 on the Paragon. The performance may change as the systems are upgraded. In the final version of this paper, we will report results for the latest versions of the operating systems in use at that time.

We are currently unable to use the logical partitioning feature of PIOFS for this application, because it is not clear to us how it can be used for arrays distributed in three dimensions. We have sought help from IBM regarding this issue. We are particularly interested in comparing the performance of logical partitioning and two-phase I/O. If we are able to use logical partitioning, we will report the results in the final version of this paper.

Acknowledgments

We thank Andrea Malagoli for giving us the source code of the application, and Ruth Aydt for helping us understand how to use Pablo.

References

[1] A. Acharya, M. Uysal, R. Bennett, A. Mendelson, M. Beynon, J. Hollingsworth, J. Saltz, and A. Sussman. Tuning the Performance of I/O Intensive Parallel Applications. In *Proceedings of Fourth Workshop on Input/Output in Parallel and Distributed Systems*, pages 15–27, May 1996.

[2] R. Aydt. A User's Guide to Pablo I/O Instrumentation. Technical report, Dept. of Computer Science, University of Illinois at Urbana-Champaign, December 1994.

[3] S. Baylor and C. Wu. Parallel I/O Workload Characteristics Using Vesta. In *IPPS '95 Workshop on Input/Output in Parallel and Distributed Systems*, pages 16–29, April 1995.

[4] R. Bordawekar, A. Choudhary, and J. del Rosario. An Experimental Performance Evaluation of Touchstone Delta Concurrent File System. In *Proceedings of the 7th ACM International Conference on Supercomputing*, pages 367–376, July 1993.

[5] D. Bradley and D. Reed. Performance of the Intel iPSC/2 Input/Output System. In *Fourth Conference on Hypercube Concurrent Computers and Applications*, pages 141–144, 1989.

[6] P. Colella and P. Woodward. The Piecewise Parabolic Method (PPM) for Gas-Dynamical Simulations. *Journal of Computational Physics*, 54(1):174–201, April 1984.

[7] P. Crandall, R. Aydt, A. Chien, and D. Reed. Input-Output Characteristics of Scalable Parallel Applications. In *Proceedings of Supercomputing '95*, December 1995.

[8] R. Cypher, A. Ho, S. Konstantinidou, and P. Messina. Architectural Requirements of Parallel Scientific Applications with Explicit Communication. In *Proceedings of the 20th Annual International Symposium on Computer Architecture*, pages 2–13, 1993.

[9] J. del Rosario, R. Bordawekar, and A. Choudhary. Improved Parallel I/O via a Two-Phase Runtime Access Strategy. In *Proceedings of the Workshop on I/O in Parallel Computer Systems at IPPS '93*, pages 56–70, April 1993.

[10] J. del Rosario and A. Choudhary. High Performance I/O for Parallel Computers: Problems and Prospects. *IEEE Computer*, pages 59–68, March 1994.

[11] D. Feitelson, P. Corbett, and J. Prost. Performance of the Vesta Parallel File System. In *Proceedings of the Ninth International Parallel Processing Symposium*, pages 150–158, April 1995.

[12] J. French, T. Pratt, and M. Das. Performance Measurement of the Concurrent File System of the Intel iPSC/2 Hypercube. *Journal of Parallel and Distributed Computing*, 17(1–2):115–121, January and February 1993.

[13] N. Galbreath, W. Gropp, and D. Levine. Applications-Driven Parallel I/O. In *Proceedings of Supercomputing '93*, pages 462–471, November 1993.

[14] W. Gropp and B. Smith. Chameleon Parallel Programming Tools User's Manual. Technical Report ANL–93/23, Mathematics and Computer Science Division, Argonne National Laboratory, March 1993.

[15] V. Herrarte and E. Lusk. Studying Parallel Program Behavior with Upshot. Technical Report ANL–91/15, Mathematics and Computer Science Division, Argonne National Laboratory, August 1991.

[16] D. Kotz and N. Nieuwejaar. File-System Workload on a Scientific Multiprocessor. *IEEE Parallel and Distributed Technology*, pages 51–60, Spring 1995.

[17] T. Kwan and D. Reed. Performance of the CM-5 Scalable File System. In *Proceedings of the 8th ACM International Conference on Supercomputing*, pages 156–165, July 1994.

[18] N. Nieuwejaar and D. Kotz. Performance of the Galley Parallel File System. In *Proceedings of Fourth Workshop on Input/Output in Parallel and Distributed Systems*, pages 83–94, May 1996.

[19] N. Nieuwejaar, D. Kotz, A. Purakayastha, C. Ellis, and M. Best. File-Access Characteristics of Parallel Scientific Workloads. Technical Report PCS–TR95–263, Dept. of Computer Science, Dartmouth College, August 1995.

[20] B. Nitzberg. Performance of the iPSC/860 Concurrent File System. Technical Report RND-92-020, NAS Systems Division, NASA Ames, December 1992.

[21] A. Purakayastha, C. Ellis, D. Kotz, N. Nieuwejaar, and M. Best. Characterizing Parallel File-Access Patterns on a Large-Scale Multiprocessor. In *Proceedings of the Ninth International Parallel Processing Symposium*, pages 165–172, April 1995.

[22] A. L. N. Reddy and P. Banerjee. A Study of I/O Behavior of Perfect Benchmarks on a Multiprocessor. In *Proceedings of the 17th Annual International Symposium on Computer Architecture*, pages 312–321, 1990.

[23] D. Reed, R. Aydt, R. Noe, P. Roth, K. Shields, B. Schwartz, and L. Tavera. Scalable Performance Analysis: The Pablo Performance Analysis Environment. In *Proceedings of the Scalable Parallel Libraries Conference*, pages 104–113, October 1993.

[24] R. Thakur, E. Lusk, and W. Gropp. I/O Characterization of a Portable Astrophysics Application on the IBM SP and Intel Paragon. Technical Report MCS–P534–0895, Mathematics and Computer Science Division, Argonne National Laboratory, Revised October 1995.