Appeared in Proc. of the 3rd Int'l Conf. of the Austrian Center for Parallel Computation with special emphasis on Parallel Databases and Parallel I/O, Sept. 1996. Lecture Notes in Computer Science 1127, Springer-Verlag, pp. 24–35.

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

Rajeev Thakur, William Gropp, and Ewing Lusk

Mathematics and Computer Science Division
Argonne National Laboratory
9700 S. Cass Avenue
Argonne, IL 60439, USA
{thakur, gropp, lusk} @mcs.anl.gov

Abstract. We present the results of an experimental evaluation of the parallel I/O systems of the IBM SP and Intel Paragon using a real three-dimensional parallel application code. This application, developed by scientists at the University of Chicago, simulates the gravitational collapse of self-gravitating gaseous clouds. It performs parallel I/O by using library routines that we developed and optimized separately for the SP and Paragon. The I/O routines perform two-phase I/O and use the parallel file systems PIOFS on the SP and PFS on the Paragon. We studied the I/O performance for two different sizes of the application. In the small case, we found that I/O was much faster on the SP. In the large case, open, close, and read operations were only slightly faster, and seeks were significantly faster, on the SP; whereas, writes were slightly faster on the Paragon. The communication required within our I/O routines 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.

1 Introduction

It is widely recognized that, in addition to fast computation and communication, parallel machines must also provide fast parallel I/O. Researchers have proposed many different types of architectures and file systems for parallel I/O, a few of which are being used in current-generation parallel machines. There is no consensus, however, as to what is the best type of parallel I/O architecture or parallel file system.

* 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.

To better understand this issue, we evaluated the performance of the parallel I/O systems of two different state-of-the-art parallel machines, with a real application workload. The two machines we considered are the IBM SP at Argonne National Laboratory and the Intel Paragon at Caltech. These machines are also the two testbeds for the Scalable I/O Initiative[2]. The application we used is a three-dimensional production parallel code developed by scientists at the University of Chicago to study the gravitational collapse of self-gravitating gaseous clouds. The application performs parallel I/O by using library routines that we developed and optimized separately for the SP and Paragon. We instrumented the I/O routines, ran two different sizes of the application on both systems, and analyzed the resulting trace files. We found that, in the small case, all I/O operations (open, close, read, write, seek) were much faster on the SP. In the large case, open, close, and read operations were only slightly faster, and seeks were significantly faster, on the SP; whereas, writes were slightly faster on the Paragon. The communication required within our parallel I/O routines was faster on the Paragon in both cases.

The rest of this paper is organized as follows. Section 2 provides an overview of related work. Section 3 describes the configurations of the two machines, the application, and the parallel I/O routines used in the application. Section 4 provides details of the experiments performed. We present performance results in Section 5 and draw overall conclusions in Section 6.

2 Related Work

We discuss related work in the area of I/O characterization of parallel applications and performance evaluation of parallel file systems.

Nieuwejaar et al. [17] performed a tracing study of all file-related activity on the Intel iPSC/860 at NASA Ames Research Center and the Thinking Machines CM-5 at the National Center for Supercomputing Applications. They found that file sizes were large, I/O 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] studied 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] report their experience in tuning the performance of four applications on an IBM SP. Del Rosario and Choudhary [9] provide an informal summary of the I/O requirements of several Grand Challenge applications.

Researchers have also studied the performance of parallel file systems. Bordawekar et al. [4] performed a detailed performance evaluation of the Concurrent

[2] See `http://www.cacr.caltech.edu/SIO/` for information on the Scalable I/O Initiative

File System (CFS) on the Intel Touchstone Delta. Kwan and Reed [15] measured the performance of the CM-5 Scalable File System. Feitelson et al. [10] studied the performance of the Vesta file system. Nieuwejaar and Kotz [16] present performance results for the Galley parallel file system. Several researchers have measured the performance of the Concurrent File System (CFS) on the Intel iPSC/2 and iPSC/860 hypercubes [5, 11, 18].

In an earlier work, we studied the I/O characteristics of a different application on the SP and Paragon [20]. For that study, we used a two-dimensional astrophysics application that performs sequential I/O (only processor 0 performs all I/O) using the Unitree file system on the SP and the PFS file system on the Paragon. In this paper, we consider a completely different three-dimensional application that is much more I/O intensive and performs parallel I/O (two-phase I/O) using the parallel file systems PIOFS on the SP and PFS on the Paragon.

3 Machine and Application Description

We describe the SP and Paragon systems, the application, and the parallel I/O routines used in the application.

3.1 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.

IBM SP. The SP at Argonne was configured as follows during our experiments. There were 120 compute nodes, each an RS/6000 Model 370 with 128 Mbytes of memory, and eight I/O server nodes, each an RS/6000 Model 970 with 256 Mbytes of memory. All 128 nodes were interconnected by a high-performance omega switch. The operating system on each node was AIX 3.2.5. IBM's parallel file system PIOFS provided parallel access to files. Each I/O server node had 3 Gbytes of local SCSI disks, resulting in a total PIOFS storage capacity of 24 Gbytes. Users were not allowed to run compute jobs on the I/O server nodes.

PIOFS distributes a file across multiple I/O server nodes. A file is logically organized as a collection of *cells*: 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.

Intel Paragon. The Paragon at Caltech was configured as follows during our experiments. There were 512 compute nodes and 16 I/O nodes, each an Intel i860/XP microprocessor with 32 Mbytes of memory. The nodes were connected by a two-dimensional mesh interconnection network. The operating system on the machine was Paragon/OSF R1.3.3. Each I/O node was connected to a 4.8-Gbyte RAID-3 disk array, and Intel's Parallel File System (PFS) provided parallel access to files. As on the SP, users were not allowed to run compute jobs on the I/O nodes.

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. 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. During our experiments, the Paragon had 16 stripe directories, and the default stripe unit was 64 Kbytes.

3.2 The Application

The application we used is a production parallel code developed at the University of Chicago. This application simulates the gravitational collapse of self-gravitating gaseous clouds due to a process known as Jeans instability. This process is the fundamental mechanism through which intergalactic gases condense to form stars. The application solves the equations of compressible hydrodynamics with the inclusion of self-gravity. It uses the piecewise parabolic method [6] to solve the compressible Euler equations and a multigrid elliptic solver to compute the gravitational potential.

The application uses the Chameleon library for communication [13], which is portable. Originally, the application also used the Chameleon library for I/O [12], but we found that the Chameleon I/O routines were not well optimized for parallel I/O on the SP and Paragon. We therefore wrote special I/O routines, described below, with the same interface as the Chameleon I/O library, but separately optimized for the SP and Paragon. The application performs all I/O via calls to these optimized routines; therefore, it 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 that are distributed in a (block,block,block) fashion among processors. All arrays fit in memory, but every few iterations, several arrays must be written to files for three purposes: data analysis, checkpointing (restart), and visualization. The application reads data only while being restarted from a previously created checkpoint. The storage order 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 values across all processors, followed by six arrays appended one after another. The arrays are stored in column-major (Fortran) order. The restart file has the same structure as the data-analysis file. The application performs all computation in double precision, but writes single-precision data to the data-analysis and restart files. The visualization data is written to four separate files. Each of those files begins with six variables (real numbers) that have the same value across all processors, followed by one array of character data. The application creates one restart file in all and new data-analysis and visualization files in each dump.

3.3 Parallel I/O Routines

Recall that, in this application, three-dimensional arrays are distributed among processors in a (block,block,block) manner. Each array must be written to a single 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, and only processor 0 actually writes data to the file. This approach is inefficient for two reasons: the sequential nature of I/O and the communication bottleneck caused by the all-to-one communication pattern.

To overcome these limitations, we wrote new routines that have the same interface as the Chameleon I/O routines, but perform I/O in parallel from all processors. Since the interface did not change, we did not have to change the application code. The new 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, the routines use the `gopen()` call for faster opens and the `M_ASYNC` mode for faster reads and writes.

In this application, the local array of each processor is not located contiguously in the file. Therefore, an attempt by any processor to read/write its local array directly would result in too many small read/write requests. We eliminated this problem by using two-phase I/O [8], 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 processors such that, in the new distribution, each processor's local data is located contiguously in the file. In the second phase, processors write their local data at appropriate locations in the file concurrently, with a single write operation each. To read a distributed array, each processor reads a contiguous block in the first phase and then redistributes it in the second phase. This method eliminates the need for several small I/O requests and also has a fairly balanced all-to-many communication pattern.

Figure 1 illustrates how our I/O routines use two-phase I/O to read/write a three-dimensional array distributed as (block,block,block). The write routine first redistributes an array from (block,block,block) to (*,*,block). In other words, after the first phase, the array is distributed along the third dimension only. In the second phase, all processors write their local data simultaneously to the file. Conversely, in the read routine, all processors first read their local data assuming a (*,*,block) distribution and then redistribute it to the required (block,block,block) distribution.

We note that the PIOFS file system also supports logical partitioning of files. A processor can specify a logical view of its local array in the global array file and then read/write the local array with a single operation, even though the local

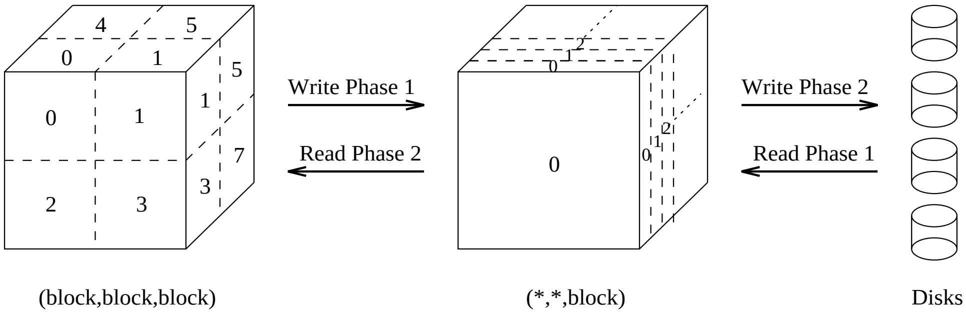

Fig. 1. Two-phase I/O for reading/writing a three-dimensional array distributed as (block,block,block) and stored in column-major (Fortran) order

array may not be located contiguously in the file. However, we found that this feature can be mainly used for arrays distributed in two dimensions (including three-dimensional arrays distributed in two dimensions). We were unable to use it in this application, because the arrays are distributed in three dimensions. Therefore, we used our two-phase I/O routines even on the SP.

The application also requires six variables (real numbers), with the same values across all processors, to be written in each dump and also read during restart. We wrote these variables by collecting all of them into a single buffer on processor 0 and then writing the buffer to the file, from processor 0 only, in a single operation. We read the variables by reading all of them in a single operation from processor 0 only and then broadcasting them to other processors.

4 Details of Experiments

To study the I/O behavior of the application, we instrumented the I/O routines by using the Pablo instrumentation library [2, 19]. We instrumented all open, close, read, write, and seek calls, and also all communication required within the I/O routines. We ran the instrumented code on both the SP and Paragon and collected trace files. The traces were visualized and analyzed by using Upshot [14], a tool for studying parallel program behavior.

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. The application is iterative, and 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 below. The I/O behavior in the remaining iterations is assumed to be similar to that in the first few iterations.

We considered 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, with all dumps performed every five iterations. The data analysis and restart files were 24 Mbytes each, and the visualization files were 1 Mbyte each.

2. **Large.** For this case, we used a $256 \times 256 \times 128$ mesh on 64 processors. Since only 120 compute nodes were available on the SP and the application runs only on a power-of-two number of processors, we could run it on a maximum of 64 processors on the SP. 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 five iterations, with all dumps performed after the fifth iteration. The data analysis and restart files were 192 Mbytes each, and the visualization files were 8 Mbytes each.

During our experiments, we did not have exclusive use of the system. To eliminate spurious results due to interference from other users' jobs and system-related activities, we ran the application several times and recorded only the run which took the least time.

5 Performance Results

We first discuss the results on the SP, followed by the results on the Paragon, and then compare the results on the two systems.

5.1 Results on SP

Table 1 shows the total number and total time for each type of I/O operation on the SP for the small case. We calculated the average time per processor as the total time across all processors divided by the number of processors. There were 200 open and close calls. The open calls were fairly expensive; close calls were not. There were 1225 seek calls that took a total time of 0.052 sec. In other words, the time for an individual seek operation was almost negligible. There were a small number of read operations during the restart. Write operations dominated the I/O, as the application is write intensive. Note that the write timings represent the time taken for the write calls to return. Data may or may not have reached the disks at the end of each write call, depending on the caching policy used by the file system. Communication for I/O (that is, the communication performed within our I/O routines) took even more time than the write operations. This was not the case on the Paragon, however, as we discuss in Section 5.2.

Table 2 shows the distribution of the sizes of individual read and write operations in the small case. Most of the reads and writes were large, since we used two-phase I/O. The few small requests were due to the reading and writing of six variables at the beginning of files. As explained in Section 3.3, these variables were read/written in a single operation by processor 0 only. The aggregate read bandwidth across all processors, computed as the total data read by all processors divided by the average read time per processor, was 48 Mbytes/sec. The aggregate write bandwidth, computed similarly, was 31.6 Mbytes/sec. We guess

Table 1. I/O operations on the SP for the small case—$128 \times 128 \times 64$ 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 2. Details of read and write operations on the SP for the small case

	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

that the file system may be using a read-modify-write algorithm to implement write operations, resulting in the lower write bandwidth.

Tables 3 and 4 show the results for the large case on the SP. The overall trend in the results was similar to that in the small case. Open operations were again very expensive. Close operations took a small amount of time, and seek operations took negligible time. The most expensive operations were communication for I/O and write. The sizes of individual read and write operations were the same as in the small case: although the mesh size was eight times larger, the number of processors was also eight times larger. The aggregate read bandwidth (33.9 Mbytes/sec.) and the aggregate write bandwidth (17.4 Mbytes/sec.) were both lower than in the small case. The lower I/O bandwidth may be because, in the large case, the ratio of compute nodes to I/O server nodes was 8 : 1, whereas, in the small case, the ratio was 1 : 1.

5.2 Results on Paragon

Tables 5 and 6 show the results for the small case on the Paragon. The counts and sizes of I/O operations were the same as on the SP. Opens were very expensive; close and seek operations were inexpensive. Communication for I/O took less time than write operations, contrary to that on the SP. The aggregate read bandwidth was 28.6 Mbytes/sec., and the aggregate write bandwidth was 17.1 Mbytes/sec. As on the SP, the write bandwidth was lower than the read bandwidth.

Table 3. I/O operations on the SP 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	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 4. Details of read and write operations on the SP for the large case

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

Table 5. I/O operations on the Paragon for the small case

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 6. Details of read and write operations on the Paragon for the small case

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

Tables 7 and 8 show the results for the large case on the Paragon. The overall trend in the results was the same as in the small case. Opens were again very expensive. The aggregate read bandwidth (33.6 Mbytes/sec.) and the aggregate write bandwidth (18.6 Mbytes/sec.) were higher than in the small case, contrary to that on the SP. The reason may be that the Paragon had more I/O nodes (16) to service requests from 64 compute nodes.

Table 7. I/O operations on the Paragon for the large case

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 8. Details of read and write operations on the Paragon for the large case

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

5.3 Comparison of SP and Paragon Results

In the small case, all I/O operations (open, close, read, write, and seek) were much slower on the Paragon. The aggregate read and write bandwidths on the Paragon were 60% and 55% of the bandwidths on the SP, respectively. However, the communication required within the I/O routines was faster on the Paragon.

In the large case, open, close, and read operations took only slightly longer on the Paragon. Seeks took significantly longer on the Paragon. On the other hand, writes were slightly faster, and communication for I/O was twice faster, on the Paragon.

On both machines, the time for opening common files from all processors was very high in both the small and large cases. We do not know the reason for the high open time, since it is related to the underlying implementation of the PIOFS and PFS file systems. On the SP, the read and write bandwidths obtained were higher in the small case, whereas, on the Paragon, they were higher in the large case, possibly because there were 16 I/O nodes on the Paragon versus only 8 on the SP.

6 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 production parallel application. We found that the relative performance of the two systems depends on the problem size. For the small case, all I/O operations were much faster on the SP. For the large case, open, close, and read operations were only slightly faster, and seeks were considerably faster, on the SP. Writes, however, were slightly faster on the Paragon. In both cases, communication for I/O was faster on the Paragon.

We note that the results in this paper are specific to the I/O access pattern of this application and should not be interpreted as a general performance comparison of the two systems. For some other access patterns, the results may be different. We also note that the results are for the particular hardware and software configurations specified in Section 3.1.

In all our experiments on both systems, we found that the time for opening common files from all processors was very high. Therefore, we recommend that parallel-file-system designers must also aim to reduce file-open time, in addition to reducing read and write times.

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 R. Jain, J. Werth, and J. Browne, editors, *Input/Output in Parallel and Distributed Computer Systems*, chapter 7, pages 167–185. Kluwer Academic Publishers, 1996.
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. 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.
9. J. del Rosario and A. Choudhary. High Performance I/O for Parallel Computers: Problems and Prospects. *IEEE Computer*, pages 59–68, March 1994.
10. 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.
11. 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.
12. N. Galbreath, W. Gropp, and D. Levine. Applications-Driven Parallel I/O. In *Proceedings of Supercomputing '93*, pages 462–471, November 1993.
13. 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.
14. 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.
15. 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.
16. 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.
17. 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.
18. B. Nitzberg. Performance of the iPSC/860 Concurrent File System. Technical Report RND-92-020, NAS Systems Division, NASA Ames, December 1992.
19. 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.
20. 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.

This article was processed using the LaTeX macro package with LLNCS style