

Parallel Programming Enhancements for Processing Hydrographic Data

Dr. M.J. Miller¹, Geary Layne¹, Doug Cronin²

¹Naval Research Laboratory Code 7440.1, Stennis Space Center, MS

²Naval Oceanographic Office, Stennis Space Center, MS

Abstract

Parallel programming techniques have been used for years to develop processing software that does the same work in only a fraction of the time. The research in this paper focuses on the I/O problem associated with a parallel application writing to a single physical disk. Included in our research are the original ideas that led to the first version of the parallel software, subsequent versions of the software derived from lessons learned from benchmark results, and speedup results of each version. The platform used was a custom built Linux Beowulf cluster running a standard Linux kernel and the MPICH parallel message-passing library. The underlying purpose of this software is to process hydrographic data having a complicated, multi-tiered format. The data processing involves reading tens to hundreds of files containing raw data, filtering out extraneous data values, and writing the filtered data to a single file used in additional processing. The problem is not computationally intensive, but bound by the system's file writing capability. Subsequent versions of the parallel software developed exploit the strengths of the system's hardware to write the output file in the most time efficient manner. Each software version uses advanced software architecture schemes to achieve better results. Results show that the more responsible the software was for organizing the data before writing, the better the speedup. The critical factor for writing data efficiently involved the limitation of writing data over a single I/O controller. Our parallel software has fantastic utility where system specifications do not allow for the use of parallel file systems, or writing data over multiple I/O controllers.

1 Introduction

The Naval Research Laboratory's (NRL) Code 7440 Production Enhancement Team at the Stennis Space Center has been tasked to develop ways to speedup hydrographic data processing at the Naval Oceanographic Office (NAVO) [Depn02]. This paper presents the final development of a parallelized version of the Pfm_loader application customized to run on a Beowulf cluster.

This paper specifically covers software development efforts in early FY02 [Sarn02]. In a series of algorithms, called Schemes D, E, and F, a parallel algorithm previously implemented in Scheme C (Figure 1) has been integrated with an improved version of NAVO's Pure File Magic (PFM) library. Former versions of this software, not mentioned, Schemes A and B, were used as a design platform for the software architecture found in Scheme C. The main goal of this work was to increase the writing rate of binned data to a physical disk. The final software version is Scheme F.

The final parallel code of Scheme F achieves the best speedup for the largest available test dataset. The simple runs (no filtering) exhibited a speedup of 10 times the speed of the original, serial algorithm. Runs with swath filtering showed a top speedup of 8. Runs with area filtering reached a speedup of 6.5. Runs with both swath and area filtering showed a top speedup of 7. In all cases, data strongly suggest that greater speedups could be achieved for larger than tested input datasets.

Section 2 of this paper is devoted to Scheme D. Section 3 presents results on an implemented threaded version of Scheme D. Section 4 presents results for Scheme E. Section 5 describes Scheme F. Detailed results are presented on the performance of Scheme F when swath and area based filtering are enabled.

Section 6 presents results illustrating the effect of the physical disk's I/O throughput rates on speedup results. Section 7 discusses results and future plans.

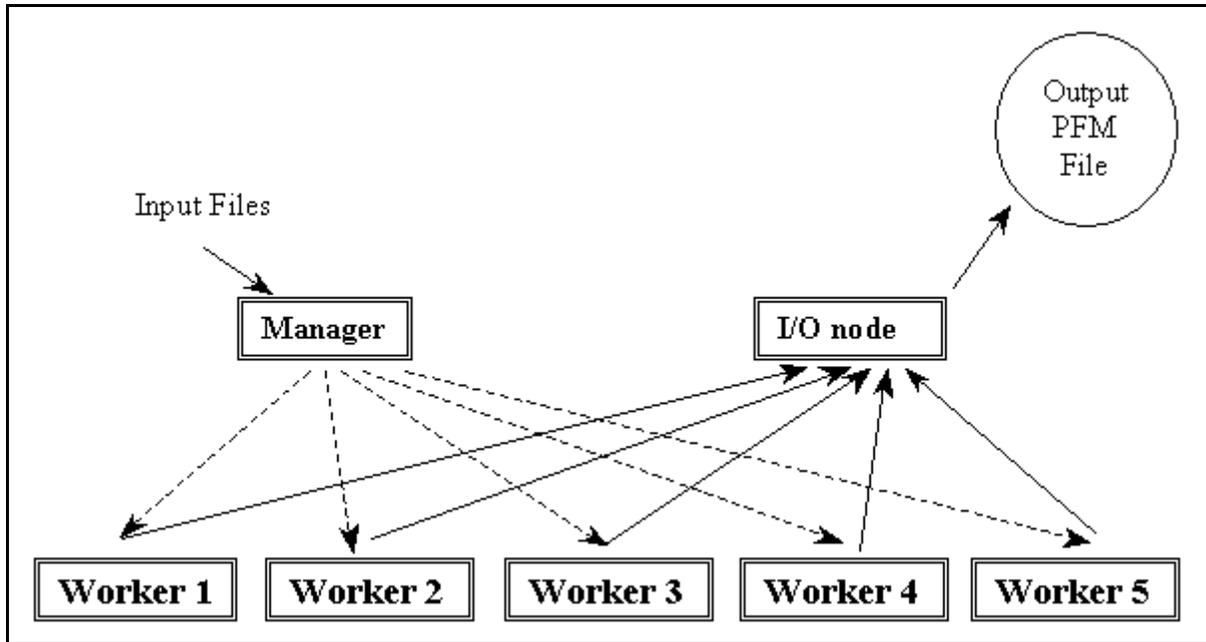


Figure 1. Parallel software design.

2 Scheme D

In Scheme D, as in Scheme C, nodes are assigned one of the following roles — a master node, an I/O node, or a slave node. More of the functionality of the original PFM library has been assigned to the slave nodes in Scheme D than in earlier schemes. The following list describes the parallel communication scheme:

- The manager node waits for requests from the worker nodes. A single file name is sent by the manager for each worker node request. The manager node terminates once the file-processing list is exhausted.
- Each worker node sends a file request to the manager node, then receives a file name to process. The worker nodes have connectivity to each file location and are responsible for reading, sorting and compressing the sounding data according to bin index in the same way as was done in the original PFM library. Once the processing file list is exhausted on the manager node, the first worker to complete processing its current file becomes identified as a grouping node.
- The I/O node receives sounding data in the PFM compressed format and writes them into the PFM-style depth blocks consisting of six sounding data and a continuation pointer. Once data from all worker nodes has been received, the I/O node performs a validity check before terminating. The PFM file is the output product allowing users to visually edit the data.

Figure 2 illustrates achieved speedups for different loads. The test datasets are denoted by the number of files run at one time (7, 12, 24, 48, 74 files). Timings have been averaged over four runs. Speedup figures are created by comparing parallel and serial runs. In the case of algorithm speedups, the comparisons are made with corresponding timings for three Message Passing Interface (MPI) processes. Speedups

exhibited by Scheme D reach about 7 for the four larger dataset loads. For the smallest dataset, Scheme D shows a speedup of around 5, cutting the run time from 2 min to 23 sec. For the largest input dataset tested, the measured speedup is 7 fold, reducing execution time from 24 min to 3 min 20 sec. The optimal number of MPI processes is 9 for this scheme.

3 Threading

3.1 Initial Plan

The overall processing flow can be improved by grouping tasks into at least two threads on the I/O node and a slave node. Tasks related to MPI communication can be accomplished by a separate thread. A testing program mixing MPI and threading confirmed that MPI Chamleon (MPICH) allows only a single thread to execute MPI calls. On the I/O node, a separate communication thread would take care of receiving buffers. The other thread would be responsible for writing data to the PFM output file. On the slave node, a communication thread would send full buffers to the I/O node. The second thread would transfer and process sounding data from input files to buffers. Tests are planned to check if the currently used dual-CPU system boards will efficiently support two application threads. A quad system board for the I/O node could be used if that proved beneficial.

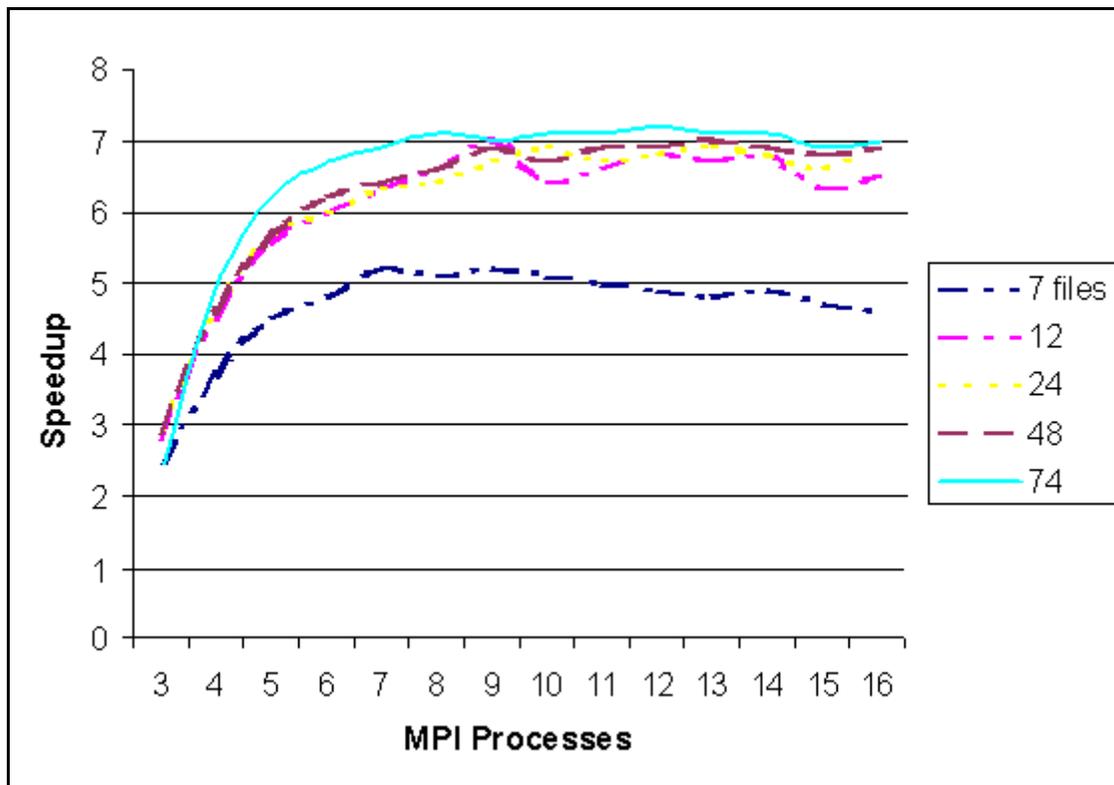


Figure 2. Achieved speedup of Scheme D for different loads.

3.2 Threading the I/O Node: Algorithm

Only one part of the above plan has been implemented: Scheme D has been tested using Linux Posix threads on the I/O node only. Processing on the I/O node has been separated into two threads. The main thread (which acts as the main communication thread) does all necessary initialization and then creates

one additional thread, called the I/O processing thread. The processing thread submits data to the output PFM file. All MPI function calls are done only by the main thread, which receives buffers from worker nodes and submits them to a work queue. The I/O processing thread checks the work queue for any full buffers and uses PFM function calls to write data to the PFM output file. An empty buffer is returned to the work queue area. The algorithm reuses buffers to avoid reinitializing buffers. All access to the work queue is safeguarded by “a mutex lock” (Mutually Exclusive Access Lock) mechanism, a standard tool available in the thread library. A small set of a few empty buffers is initialized in advance on the I/O communication side. If all empty buffers on the I/O communication node are used, that thread enters the work queue and receives their empty buffers. If no empty buffers are available in the work queue area, the I/O communication thread initializes a fresh buffer. If initializing the fresh buffer fails, the I/O communication thread waits for the I/O processing thread to return a buffer. The creation of additional buffers is always expected because PFM library operations are the slowest part of processing (since these library operations require multiple accesses to the physical disk drive).

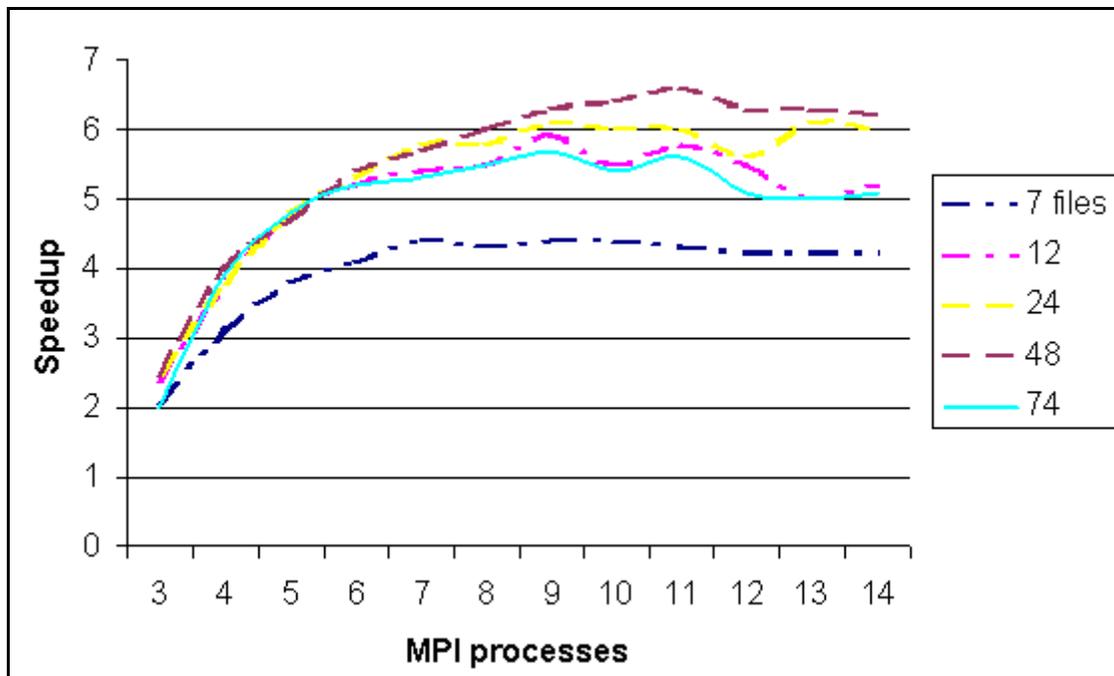


Figure 3. Achieved speedups of Scheme D threaded for different loads.

3.3 Threading the I/O Node: Results

Figure 3 illustrates achieved speedups for different loads. Speedup values for the Scheme D-Threaded version are noticeably smaller than the original Scheme D. Such results could be attributed to the additional overhead of the thread library. However, results for the largest test dataset (74 files) are especially disappointing, due to the lack of control of the memory usage on the I/O node in the threaded code. This preliminary threaded version has no control on the number of buffers used to keep incoming buffers on the I/O node. Since worker nodes deliver buffers much faster than the I/O node could possibly write them to a (slow) physical disk, incoming buffers forced the operating system on the I/O node to use disk swap space, causing a significant slow down in processing.

4 Scheme E

4.1 Grouping Sounding Data into Packages of six

The improved version of the PFM library, as well as the original library, writes data to the physical disk in fixed length blocks. Each block can hold up to six sounding values (the value of six is configurable). To take advantage of these blocks, sounding data are sent in groups of six (with the same bin index). At first, slave nodes send depths grouped into packets of six (with same bin indexes) to create full depth records in PFM style. When all input file names have been distributed (and most of the files have been already processed) the finishing slave nodes have some leftover depth data. Separately, each node cannot create a final full depth record containing 6 sounding values for one bin index. Under the direction of the master node, the first slave node to finish processing becomes a sorting and grouping node. Then, the other slave nodes send their leftover data to that sorting node for consolidation. Subsequently, full records are sent to the sorting and grouping node (the one which currently is accepting data) with the final leftover data sent to the I/O node. This additional effort of sorting into groups of six depths has the advantage of increasing the speed of writing the data to disk as well as avoiding any need to reread and rewrite partially empty depth records. Further improvements include a new interface function to the PFM library to accommodate direct writing of blocks to the PFM file. Also, slave nodes are given the additional task of creation of complete depth blocks. This improvement reduces the I/O node's task to simply updating the continuation pointers before writing data to disk.

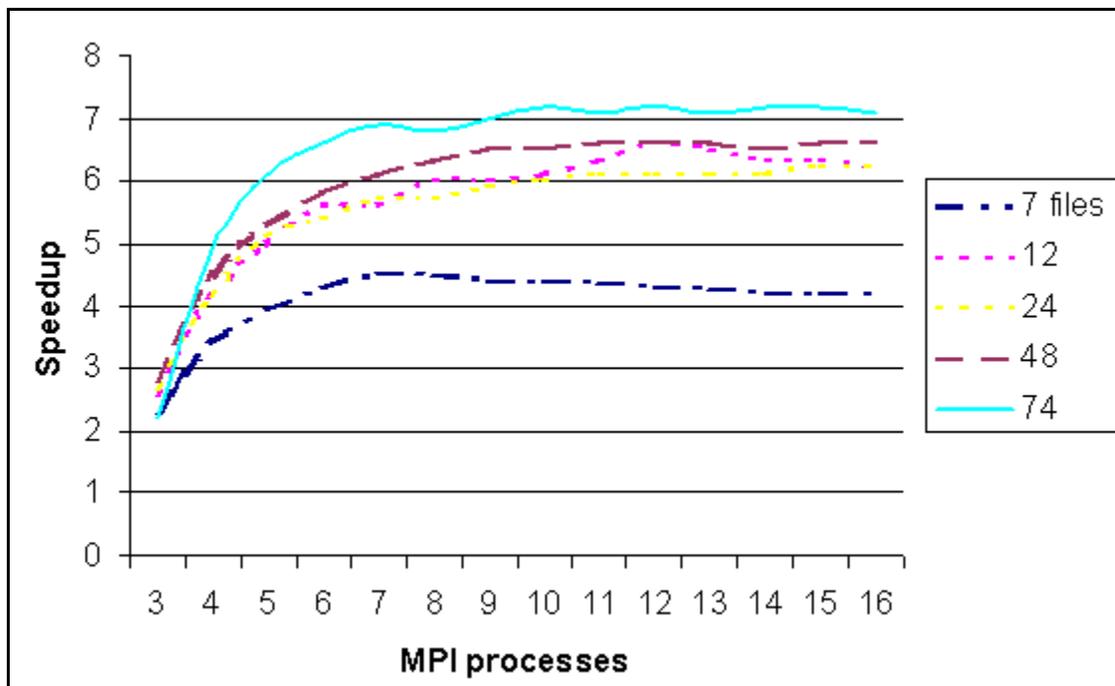


Figure 4. Achieved speedup of Scheme E for different loads.

4.2 Results

Figure 4 illustrates achieved speedups for different loads, compared with the original serial application. The timings have been averaged from four runs. A comparison between speedup numbers for Schemes D and E shows that values for Scheme E are noticeably smaller, except for the largest test dataset. For the smallest dataset, Scheme E shows a speedup around 4.5, reducing the run time from 2 min to 26 sec. For

the largest input dataset tested, the measured speedup is around 7, reducing execution time from 24 min to 3 min 19 sec. The optimal number of MPI processes is 10 for larger datasets in Scheme E.

4.3 Filtering, Recomputing Steps and Final Tune-up

The original filtering functionality works well in the Beowulf cluster environment. The original swath filtering was being handled separately for each input file. Thus the new distributed scheme of processing input files by a group of slave nodes has no effect on swath filtering. The same has been found for the recomputing step and area based filtering, which run without any changes to the code because they are executed in the serial phase, on the I/O node, only after the PFM file has already been created.

5 Scheme F

A tuned version of the new PFM library was used in Scheme F. Scheme F was tested with four different setups, involving two possible filtering procedures: swath filtering, area filtering, both swath and area filtering, and no filtering (called “simple runs”). Each of these four setups has been tested with the standard five data test loads. Results are presented in figures 5 – 7.

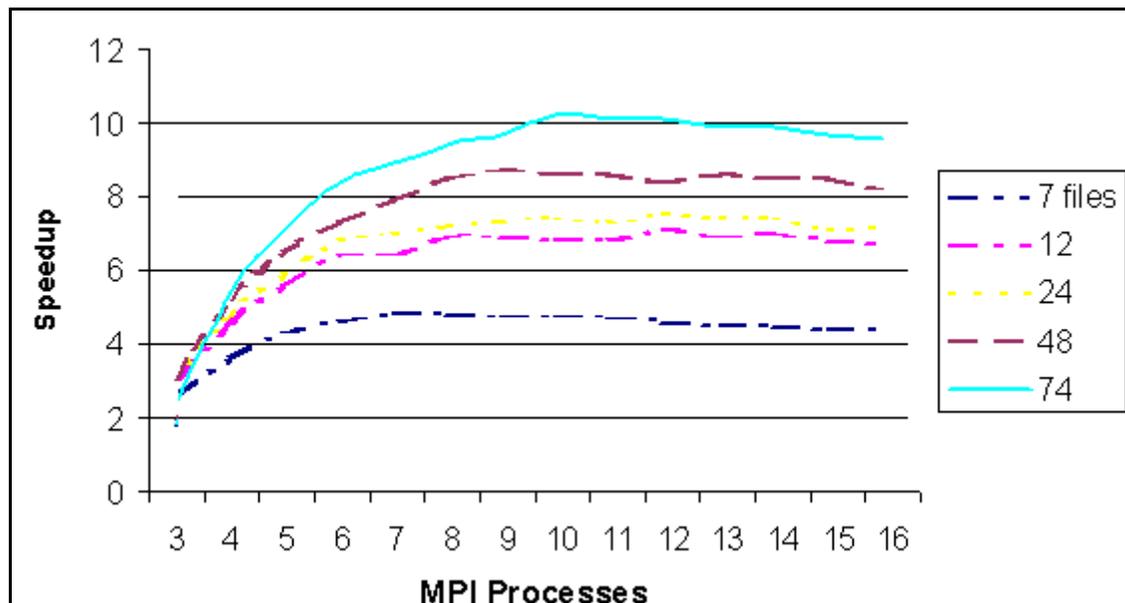


Figure 5. Achieved speedups of Scheme F with no Filtering.

It was found that Scheme F achieves the best speedup for the largest available test dataset. The simple runs (no filtering) exhibited a speedup of 10. Runs with swath filtering enabled showed a top speedup of 8. Runs with area filtering reached a speedup of 6.5. Runs with both swath and area filtering enabled showed a top speedup of 7. In the simple runs, swath filtering and area-based filtering are turned off. Figure 7 illustrates achieved speedups for different loads and filter processing. For the smallest dataset, when swath filtering is turned on, worker nodes filter the sounding data by swath before assembling them and sending to the I/O node. Since swath filtering puts additional processing onto worker nodes, such code behavior is to be expected. Worker nodes perform swath filtering on sounding data read from the input files. When area filtering is also turned on, the I/O node also performs area filtering of sounding data. Since area filtering is done exclusively by the I/O node after all sounding data has been written to the PFM file, the area filtering is performed in the serial phase of overall processing.

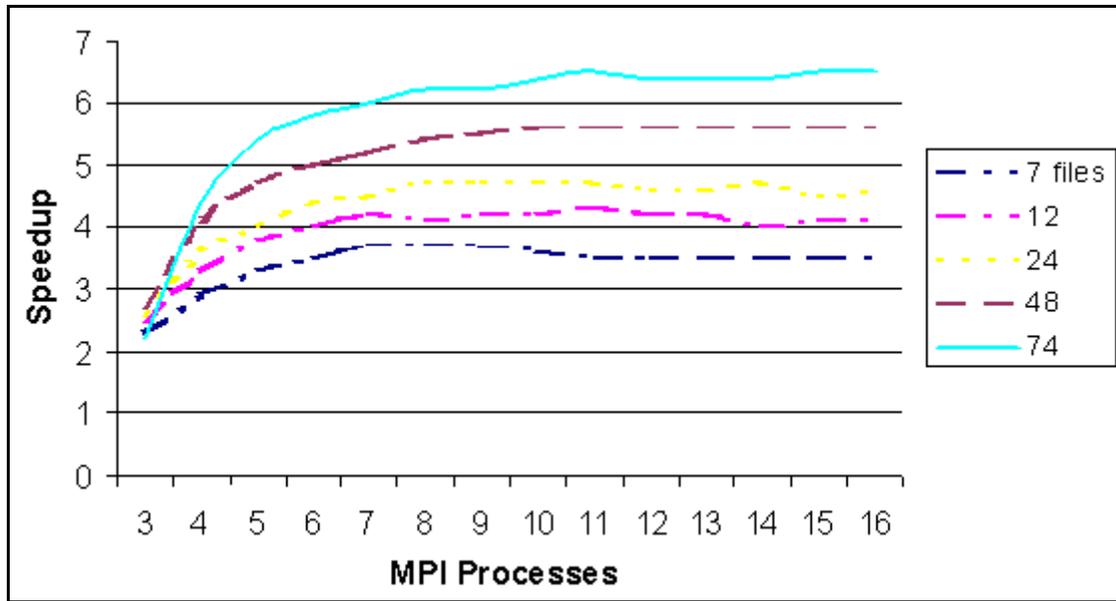


Figure 6. Scheme F with Area Filtering.

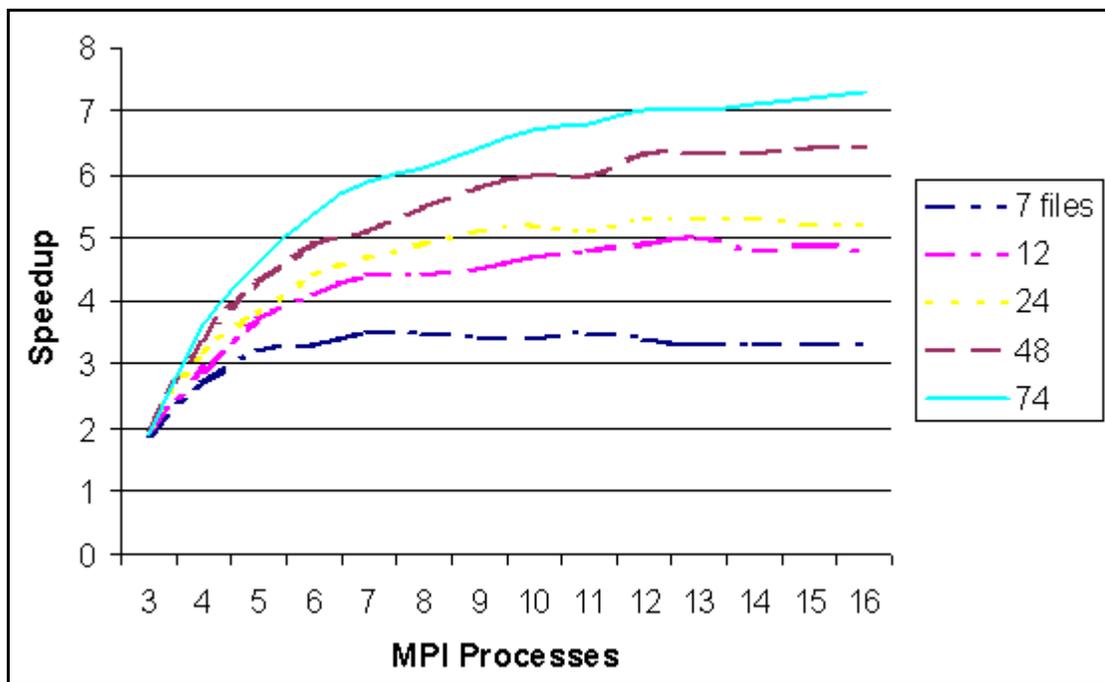


Figure 7. Scheme F using both filters.

6 Final Version

Figure 8 illustrates achieved speedups for different loads. The latest datasets are denoted by the number of files run at one time (100, 200, 300, 400, 500, 600 files). Timings have been averaged over four runs. Speedup figures are created by comparing parallel and sequential runs. For the smallest dataset, a

maximum speedup of 3.8 is achieved when using 8 processors, cutting the run time from 2 min to 32 sec. For the largest input dataset tested, the measured speedup is 12.6, reducing execution time from 33 hours to 2 hours 39 min. The optimal number of processors is 16 for this scheme.

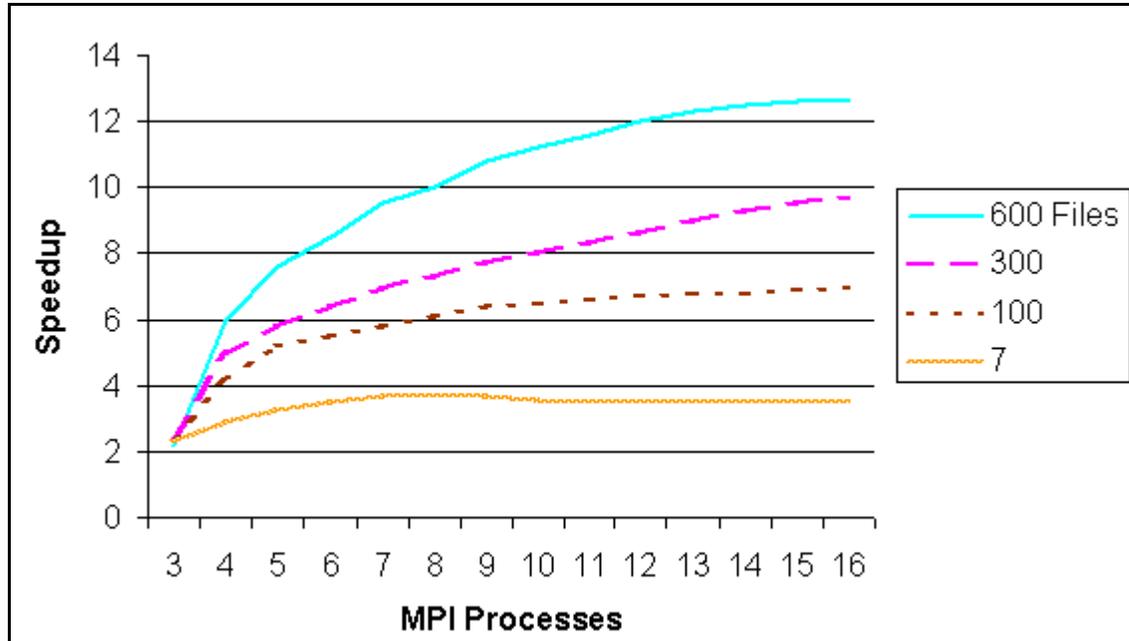


Figure 8. Final version speedup.

7 The Role of Hard Drive Performance

This section presents arguments to support the claim that the application Pfm_loader is I/O bound. When Scheme D was developed, upgrading the BIOS on each Beowulf cluster node resulted in significant improvement in the throughput of cluster IDE (ATA 100) disks. These changes significantly affected the run time of parallel codes as well as the original serial code. The changes for the original serial code are in the range from 5% to 11% (table 1). The resulting speedup is presented in figure 8. Speedup numbers are derived by comparing parallel and serial runs. In the case of algorithm speedups, the comparisons are done with corresponding timings for three MPI processes. The changes range from 29% to 57%, which at least triples that of corresponding percentage changes for serial runs.

Table 1. Changes in hard drive performance.

7	Number of Generic Sensor Format (GSF) input files				74
	12	24	48		
Timings of serial code, before BIOS upgrade					
125.106	203.57	439.253	946.034	1607.132	
Timings of serial code, with BIOS upgrade					
118.439	192.216	412.311	893.726	1433.211	
Percentage change					
5.33%	5.58%	6.13%	5.53%	10.82%	

8 Summary

The optimal number of MPI processes for the simple runs with large datasets is 16. When area filtering is turned on, the I/O node filters sounding data by geographic area. Since area filtering is done exclusively by the I/O node after all sounding data has been already written to the PFM file, this processing adds to the length of the serial phase in the overall processing. The serial processing nature of area filtering causes the greatest reduction in performance. The final parallel code, version 1.0, achieves the best speedup for the largest available test dataset. The simple runs (no filtering) exhibited a speedup of 12.6 times the speed of the original, serial algorithm using 16 processors. In all test cases, performance data strongly suggest that greater speedups could be achieved if the Beowulf cluster is attached to a data center configured with a parallel file system. The parallel file system will allow the output PFM file to be written using multiple I/O controllers simultaneously.

9 Acknowledgements

This work was sponsored under Program Element 0603704N by the Oceanographer of the Navy via SPAWAR PMW 155, Captain Bob Clark, Program Manager. The Naval Research Laboratory (NRL) wishes to acknowledge contributions the Naval Oceanographic Office (NAVOCEANO) has made to this research. In particular, NRL thanks Mr. Jim Braud, Mr. Jan Depner, Mr. Dave Fabre, Mr. Roger Meadows, Mr. Dave Richardson, Ms. Barbara Reed, and Mr. Steve Nosalik for contributing their ideas, test data, and original software sources. Additionally, the authors wish to acknowledge contributions made by the University of Southern Mississippi Hydrographic Science Research Center located at Stennis Space Center.

10 References

- Depner, J. et al, *Dealing with Increasing Data Volumes and Decreasing Resources*, Oceans MTS/IEEE, 2002.
- Miller, M.J., Layne, G., Sarnowski, C., Fabre, D., *Parallel Programming Enhancements for Processing Hydrographic Data*, IEMS conference, Cocoa Beach, FL, 2003.
- Sarnowski, K., Miller, M.J., Layne, G., *Project Report FY02*, Hydrographic Research Center, 2002.