FINAL REPORT

High Performance Input/Output for Parallel Computer Systems

CESDIS project number 5555-20

W. B. Ligon, Principle Investigator

HIGHLIGHTS

- Developed TPAW parallel system simulator for studying parallel architecture performance
- Developed PVFS parallel file system including several distinct user interfaces useful for target applications
- Parallelized AVHRR calibration and navigation code.
- Parallelized PNN image classification code.
- Developed out-of-core numerical routines
- Studied algorithm performance on TPAW, Clemson DCPC, and CESDIS Beowulf
- Explored using parallel I/O for distributed objected-oriented database

PROJECT GOALS

The goal of our project is to study the I/O characteristics of parallel applications used in Earth Science data processing systems such as Regional Data Centers (RDCs) or EOSDIS. Our approach is to study the runtime behavior of typical programs and the effect of key parameters of the I/O subsystem both under simulation and with direct experimentation on parallel systems. Our three year activity has focused on two items: developing a test bed that facilitates experimentation with parallel I/O, and studying representative programs from the Earth science data processing application domain. The Parallel Virtual File System (PVFS) has been developed for use on a number of platforms including the Tiger Parallel Architecture Workbench (TPAW) simulator, The Intel Paragon, a cluster of DEC Alpha workstations, and the Beowulf system (at CESDIS). PVFS provides considerable flexibility in configuring I/O in a UNIX-like environment. Access to key performance parameters facilitates experimentation. We have studied several key applications from levels 1, 2 and 3 of the typical RDC processing scenario including instrument calibration and navigation, image classification, and numerical modeling codes. We have also considered large-scale scientific database codes used to organize image data.

The stated goals for this projects are as follows:

- Develop an understanding of the performance potential of I/O architectures with respect to the requirements of EOS applications,
- Provide a mechanism to integrate this understanding into the RDC or DAAC processing environments at NASA,
- Conduct an evaluation of parallel I/O architectures relative to the requirements of parallel applications, and
- Integrate the results of this study into the IIFS test bed being developed by NASA Code 930.1.

Our efforts have focused on three issues:

1) Developing an experimental environment for the study,

2) Studying the performance of several applications, and

3) Integration with NASA GSFC Code 935 IIFS software.

Our experimental environment is to consist of two major components. The first and primary one is a simulation environment for studying the effects of various architectural parameters on I/O performance. For this component we are building on existing infrastructure in the Reconfigurable Architecture Workbench, which was developed under a previous research program. The second is small-scale parallel computing system utilizing clusters of high-performance workstations built through an equipment grant from the NSF. This second facility is much less flexible in its capabilities, but does provide a means for validating some of the results obtained under simulation. This second approach also provides a view of the potential to be had in low-cost systems for Earth Science Computations.

Another critical component of our experimental environment is the means to configure and access a complex parallel I/O subsystem from a parallel application. To this end we have developed the Parallel Virtual File System (PVFS). PVFS controls access to files that are distributed across multiple I/O nodes in a parallel system by multiple compute nodes in the system. PVFS is not a true file system in that it uses the native file system of the host operating system for disk block allocation and management. PVFS manages only those aspects that pertain to parallel files and parallel access to a file. In addition to providing simple file access, we have explored a number of different interfaces to a parallel file system including a new type of interface called the scheduled I/O interface. Because PVFS sits on top of a standard Unix system interface, it is easily moved between a number of host platforms and parallel system simulators.

In our study we consider parallel systems as a collection of processors each with their own local memory and a network that provides basic message passing capability. This network can be a simple as an Ethernet, or as complex as cross bar, torus, or hypercube network. I/O in these systems are provided by some number of I/O processors, where each I/O processor is a processor with local memory and directly attached storage devices such as disks and tape drives. I/O processors can be either dedicated or can also be used for computation. The number of I/O processors can be the same or different than the number of compute processors. I/O devices are attached to the I/O processors via a device bus (such as a SCSI bus) and a device controller. One or more devices can be connected to the device bus and one or more device controllers can be attached to a given I/O processor.

Our performance models include device behavior (focusing on Winchester disk and digital audio tape (DAT) devices), I/O bus performance, and interconnection network performance. The load placed on these facilities is determined by the behavior of software both at the system and application levels. System software consists primarily of device driver codes, message passing libraries, and file system software. Of these, the parallel file system code is unique to this system model. In addition, key design choices in the message passing software may have an impact on I/O performance. Key issues include the amount of data copying performed and details of the networking protocol.

The main focus of our application study is on processing level zero telemetry data to levels 1 and 2, data product generation, and automatic metadata extraction. Level 1 and 2 processing algorithms include sensor calibration, georegistration, correction and enhancement. Data product generation is highly application specific, but would include such things as vegetation index, snow and ice cover, sea surface temperature, atmospheric ozone content, etc. Metadata extraction is the process of preparing data sets for inclusion in an earth science database by generating browse products and summary data. These activities would comprise a large share of the constant processing requirements for a typical RDC or DAAC scenario in both preprocessing and reprocessing modes. Additional applications include out-of-core numerical methods used in processing high-level general circulation models (GCMs) and distributed object-oriented database codes used to organize and access Earth science data in and RDC or DAAC.

In order to explore integration of our results with ongoing efforts of NASA/GSFC Code 935 we have built an RDC consisting of a GOES satellite ground station, a remote sensing data product database, a mass storage system, and a Beowulf cluster of 18 Pentium PCs. We are expecting to add an HRPT ground station in the near future. Using these facilities we are experimenting with storing data received at the ground station in a parallel file system on the beowulf cluster, and issuing processing requests to the beowulf cluster for parallel execution. The object database is maintained on the existing RDC platform, although we are considering a parallel implementation in the near future.

EXPERIMENTAL ENVIRONMENT

Development activities for our experimental environment included the development of the TPAW simulation environment and the PVFS parallel file system. In addition, these systems were adapted to the study of parallel architectures based on clusters of workstations (COWs) or piles of PCs (POPCs) such as the Clemson Dedicated Cluster Parallel Computer (DCPC) and the CESDIS Beowulf architecture. The following sections describe these development activities.

TPAW Simulation Environment

A key drawback to RAW as a tool for studying I/O systems is that RAW'sprocessor simulator uses instruction-level simulation, which is to saytarget programs are compiled to an abstract assembly language an interpretedby the simulator. This was designed as such in order to facilitate thestudy of reconfigurable processing elements in a previous research program. In order to study I/O system, it is important that considerably longer programs that are practical to study with such a system (due to the long simulation time) are used. Since processor instruction set is not a key issue in this study, the processor simulator has been replaced with a new module that executes the target application compiled to binary code suitable for the host on which the simulation is to be run. This results in a simulation that is several orders of magnitude faster than under the old system. Available systems that utilize such a technique are limited in that they only work for one host processor type and are limited to MIMD, and in some cases only shared memory architectures. In order to maintain RAW's flexibility in these areas, a new simulation tool was developed. This tool uses a source-code augmentation technique that maintains a high degree of portability.

All simulator code is written in C, and few vendor-dependent system calls are used. Where possible development utilized POSIX compliant calls. This system simulates both SIMD and MIMD architectures and focuses on message passing systems (though shared memory is supported as well). Control and Network modules from the RAW simulator transfer readily to the new platform and the PVFS file system and I/O device models have been developed with the new simulator in mind. The details of TPAW have been documented in a Masters Thesis by Mr. R. Agnew and a paper co-authored by Mr. Agnew and the Principle Investigator is to be submitted to an international journal in September, 1994.

PVFS - Parallel Virtual File System

There are four purposes behind the design and implementation of PVFS. First, the system is designed to avoid common bottlenecks. TCP connections are used to pass data instead of a message passing library. Files used to store the parallel data are mmap()'d or accessed with low-level UNIX read() and write() calls. Second, the software is designed to be portable. While the system will not run in a heterogeneous environment, the software was written using standard UNIX system calls. Currently the software will run on both the DEC Alpha running OSF/1 and the Intel 80x86 processor running Linux. Third, the system is designed with a familiar user interface. The calls used to open, close, read, and write parallel files mimic their UNIX counterparts, and modifications to a file's metadata are made with a call similar to an ioctl(). Because of this many applications can be made to take advantage of PVFS simply by changing the file I/O calls in the program to the PVFS versions. Finally, the system is designed to be flexible. The user has a great deal of control over the distribution of a file across the partitions in the file system and the view that an application has of this file. PVFS consists of four major components: the pvfsd, the pvfsmgr, the iod, and the library of calls used by applications. Each of these will be discussed in turn.

The PVFS Daemon

The purpose of the pvfsd is twofold; it serves as a link between applications on a single machine and the pvfs manager (pvfsmgr), and it provides information on mounted parallel file systems to these applications. The pvfsd runs on all machines where applications might be run. It stores data on mounted file systems and their associated managers. It also passes on requests to mount or unmount file systems, and open, close, or unlink files, to the manager. The pvfsd accepts connections from applications at a specified port. Applications use library calls to connect to this port and make requests using only the full path to a parallel file. No information about the location of the file on other machines is necessary; all mapping is handled by the system.

When a request is made to mount a parallel file system, the pvfsd will first determine the address of the appropriate manager. This is done using information passed on by the mount program if available. If not, a file similar to the fstab file used by mount is searched for an entry corresponding to the mount point. It then connects to the correct manager and requests that the file system be mounted. If successful, the address of the manager is cached along with the name of the file system.

When a file is opened, the pvfsd determines the file system being accessed by matching part of the path to the file with a pvfs mount point. Once a match is made, a connection is re-established with the manager, and the addresses of the I/O daemons for that file are passed back from the manager to the application. In addition, a file ID is also returned. The file ID is used by the application when referring to a parallel file instead of the filename. At this point, the application can directly access the I/O daemons via the addresses returned from the PVFS manager.

The PVFS Manager

The pvfs manager (pvfsmgr) is responsible for all functions related to iods. This includes starting and stopping the iods on remote systems when necessary, and servicing file access requests (except read and write) from applications by passing the necessary commands on to the appropriate iods. In order to keep track of the iods associated with a given file system, the iodtab file is used.

The iodtab file is similar to the fstab file used by mount. For each file system managed, a set of permissions, a path to the metadata file, and a list of iods is kept in this file. For each iod, the path to the parallel files themselves is also recorded. This path may be different for each iod listed for a file system, and an iod may be listed any number of times, both for a given file system and for other file systems. This allows multiple iods to run on the same machine, possibly serving parallel files off of different disks or serving separate parallel file systems altogether.

When a request to mount a file system is made the pvfsmgr will start an iod on each machine listed for that file system. Only if all iods are successfully started will the mount itself complete. Otherwise, an error will be reported. Requests to open a file result in messages being passed to all the iods from the pvfsmgr to open the parallel file. If all of the iods successfully open the file, a file ID is passed back to the application (through the pvfsd) along with the addresses of all the iods involved. The pvfsmgr will again be contacted when the file needs to be closed. It will then report this to the iods.

Metadata

It is the responsibility of the PVFS manager to distribute the metadata describing a file when it is opened. All metadata pertaining to files on a file system is kept in a single file. This metadata file describes the distribution of files across the partitions in a parallel file system. It contains the standard information on a file, the userid of the owner and permissions, as well as additional information specific to a parallel file system. This information includes:

- number of nodes (partitions) the file is spread across
- the base node
- the stripe size

These parameters are used by both the iods and the application to determine the location of data in a file. For each physical partition in a file system an iod is running whenever the file system is mounted.

The I/O Daemon

It is the responsibility of the I/O daemon (iod) to handle all file I/O for its partition. This includes reading, writing, creating, and removing parts of the parallel files contained on its partition. The iod is started by the pvfsmgr when a file system is mounted, who passes on information telling the iod about the file system for which it will provide service. Requests to read or write a file are sent directly to the iod from the application, and use a file ID, not the parallel file name. This file ID is obtained by the application from the pvfsmgr when the file is opened. The iod uses memory mapped I/O to speed file access when possible. This memory mapped region grows and shifts with file access, and if the file is opened for writing an area at the end of the file is preallocated in expectation of a growing file. This is especially useful when a file is initially created, as it eliminates extraneous mapping during file writing. The maximum size of this window can be set to prevent the iod from hogging too much memory. Using the metadata for a parallel file, the iod takes a request from an application to access data and maps that access to the data on its partition. In the case of a read request, this data would then be returned to the application through the TCP connection. In the case of a write, the data sent from the application would be written in to the correct locations in the parallel file based on this mapping.

The library routines used by the application are responsible for sending the correct data to each iod (in the case of a write), or reading the data from the iods back in the correct order (in the case of a read).

User Interface

PVFS gives the user a great deal of flexibility with regards to the distribution of a parallel file as well as allowing the user to set a logical "view" of the file. Each file in the parallel file system is striped across the partitions with a user-defined stripe size. This allows the user to tailor the stripe size to match characteristics of the data in the file. In addition, a separate "stripe offset" is allowed as well. This defines a point in the file where the striping should start. This feature allows the user to easily map files with header blocks, which would normally destroy the clean mapping of data into stripes.

Users may also want to view only certain parts of a file. PVFS allows the user to specify a stripe size, stride, and offset that define a logical view of the file. Unlike most other parallel file systems, this logical view is completely independent of the physical partitioning of the data.

The PVFS system call library

A library of calls is available to application programmers in order to make pvfs easy to use. Basic file access routines (open, close, read, write, ioctl, lseek, unlink)are all available. This should make it possible to switch to pvfs by simply dropping in pvfs routines in the place of standard UNIX calls.

The pvfs_ioctl call is used to get and set parameters relative to an open parallel file. This includes the current "view" of the file from the user's point of view, as well as the parameters that define how the file is distributed across partitions on the parallel file system.

At the same time, the UNIX file system interface [RI78], which has been a standard for a number of years, is proving to be difficult to use and inefficient as a parallel application interface for a number of reasons. The difficulties in programming with this interface arise from a number of areas:

- Multiple accesses are needed to access multidimensional data
- Explicit seeks are needed to access partitioned data
- External synchronization is needed to manage access to shared data

These difficulties in turn lead to two types of inefficiencies. First, additional overhead is caused by the number of system calls needed to perform the necessary seeks and accesses for partitioned or multidimensional data sets. Second, caching and prefetching by the file system is often impaired by the access patterns of these applications, which often do not match the patterns of sequential applications.

New interfaces and mechanisms for I/O accesses are being developed in order to address these problems, including methods to describe I/O access patterns, support for complex data types, and collective I/O. It has been shown that these mechanisms can be effective for parallel I/O in at least some environments but it is not clear how these interfaces impact application development and performance in many environments, and especially in a distributed network environment. There are examples of parallel I/O systems that have not performed well for some applications, even on the machine they were designed for.

We have developed several parallel I/O interfaces in response to perceived needs encountered while developing parallel out of core applications. In examining these interface options, we have also developed a new paradigm for applying collective I/O in SPMD programs which we call scheduled I/O. Scheduled I/O promises to provide the following features:

- Collective I/O without implicit synchronization,
- Managed prefetching and caching,
- Data-flow synchronization support for shared data, and
- Irregular and unbalanced I/O support.

By providing such a set of primitives and a simple method for describing accesses to the IODs that make up the parallel file system, we have made developing new interfaces for PVFS relatively simple. Thusfar a UNIX-style interface (described above), a multi-dimensional block interface, and two scheduled I/O interfaces have been developed. The UNIX-style interface provides extensions to allow for strided accesses by partitioning a file. The last three interfaces are described below.

Multi-dimensional Block Interface

The PVFS block interface is designed to help in the development of out of core algorithms operating on multi-dimensional data sets. It allows the programmer to describe an open file as an N dimensional matrix of elements of a specified size and partition this matrix into a set of blocks. Blocks can then be read or written by specifying their indices in the correct number of dimensions. The dimensions and position of the block are used to create a single request to access the block stored on the file system. Accesses are made independently by all processes, so no constraints are placed on order of access and there is no implicit synchronization.

In the two dimension case, accesses are converted into a strided access. This allows the entire block to be read with a single request. When the data is defined to be of more dimensions, a batch request or nested strided request mechanism must be used. In all cases, the data in the file is currently stored in row major matrix order.

Scheduled I/O

Many types of applications have a regular I/O access pattern. For example, histogram equalization, calibration and navigation, and a number of other image processing algorithms process image data on a line-by-line basis. In a parallel implementation, scan lines are often doled out in a round-robin or block fashion. One approach to improving performance in this situation is to use collective I/O.

A collective-I/O interface is one in which all compute processes cooperate to present a single request to the file system, retaining the information that the individual requests make up a whole and allowing the file system to use this information to provide better performance. An obvious choice for a programming model for applications using collective I/O is the SPMD model. An inherent problem with many implementations of collective I/O is the necessity for an implicit synchronization at the point where I/O must take place or a message being passed from each compute process. However, if I/O for a parallel application is done in a regular pattern, then a single process should be able to provide the description for the entire group with a single request. This reduces the number of control messages and control connections while at the same time providing a logical grouping of the accesses. In addition, if this process is designed appropriately and not directly involved in all the data transfer or computation, it is free to schedule this I/O ahead of time. This allows request processing and data transfer to the compute processes to overlap computation.

Our scheduled I/O design uses this approach to provide the following advantages over typical collective I/O implementations:

- No strict boundary,
- All I/O specified with a single call, allowing IODs to optimize disk access, and
- I/O request and data transfer can be overlapped with I/O and computation

Instead of synchronizing, all compute processes simply pass a message to the scheduler informing it that they have completed the access; it is unnecessary for them to wait for all processes to finish the I/O access in order to begin computation with current data.

Scheduled Block I/O

In applications with regular access patterns to a multi-dimensional data set, it makes sense to provide a means for describing these access patterns and simplifying the process of writing programs that access this data. The scheduled block I/O interface is an attempt to provide a mechanism for writing SPMD programs that operate on multidimensional data sets distributed to compute processes in a row, column, checkerboard, or broadcast manner. The matrix dimensions are specified using the same block I/O interface, and the distribution mechanism is specified for each open file descriptor. Blocks of the matrix can then simply be read or written, and a mapping function, transparent to the application programmer, moves the file pointer to the beginning of the next block after each access.

Instead of using a set of parameters that specify a partition of the matrix, a set of mapping functions are implemented for each distribution. These functions are used to reposition the file offset based on the number of compute processes accessing the file in tandem and the distribution mechanism being used for the file. New types of distributions can be implemented simply by adding the appropriate mapping functions to the interface.

As in the scheduled I/O interface, a separate process schedules the accesses to the file for the compute processes, except that a batch mode is used to specify to the IODs the set of accesses for each process. The accesses are then organized by the IODs to optimize disk access patterns before the data is transferred.

APPLICATIONS

The key characteristic of this project has been studying the I/O behavior of real-life applications. The applications used in this study were taken from different levels of the typical RDC/DAAC processing cycle and include both image processing routines and database search routines. In each case the applications were taken and run in their entirety. In several cases the codes were partial rewritten to improve their I/O characteristics. This approach was taken because in many cases the original programmers had not considered I/O performance but were focusing on the program's functionality. Each application was studied using both parallel and sequential I/O systems. The following sections briefly describe each program studied.

AVHRR Calibraion / Navigation

The processing algorithms we expect to comprise the bulk of a typical DAAC scenario would consist of those algorithms that process raw instrument telemetry data into specific data products needed by the scientific community. Some of these algorithms would be run as a standard processing suite during data ingest, others would be the result of a specific request for data. In each case, large amounts of data would need to be output to and input from archival storage, in addition to transfers between secondary staging devices and main memory. During the last year we have established contacts with NASA Code 930.2 (the International Data Systems Office) and have been working with them on codes for calibration and registration of radiospectroscope data, specifically AVHRR. We expect to continue with algorithms for standard data products such as vegetation index and sea surface temperature. This group is also looking towards the MODIS sensor as a natural follow-on to AVHRR. These algorithms are representative of those used on other similar radiospectroscopes, and could be adjusted to account for different spatial and spectral resolutions. We believe this is a good start at looking at critical and well-proven algorithms. In the future we hope to focus on more exotic and experimental algorithms.

PNN Satellite Image Classifier

This program uses a probabilistic neural network to perform classification on a multi-spectral satellite image. This routine uses established ground truth to create a set of training vectors that are then compared against image pixels in order to classify them for land cover/land use or to identify image content such as cloud cover. This routine can be used early in the processing cycle to extract image content vectors for inclusion in the metadata to enable content-based search. Alternatively, this routine can be used later in the processing cycle to identify any number of image content features. This routine can also be used as a precursor to more sophisticated classification methods by performing a fast classification used to define regions of interest for slower but more powerful classification routines. This particular classifier is relatively quick, and thus is more dependent on I/O performance.

Out-of-Core Numerical Codes

High level Earth science codes include numerical models of Earth systems including atmospheric models, ocean models, and Earth magnetic field models. At the heart of many of these applications is the solution of systems of linear equations. We have worked with a number of codes designed to model large electromagnetic fields using Method of Moments techniques that relay on large dense matrix equations. In order to process some of the larger codes out-of-core techniques are required on many of the more cost-effective parallel architectures due to memory constraints. We have developed a number of parallel out-of-core numerical codes based on matrix multiplication, matrix-vector multiplication, and matrix decomposition including Gaussian elimination, Jacobi and Gauss/Siedel iterative methods, domain decomposition, and relatively new reduced current fidelity techniques. These codes have been tested within a number of parallel numerical applications related to Earth science.

Object Databases

One of the least understood applications areas in RDC/DAAC processing is the area of intelligent data management. Of primary interest is the creation and access to an object database that records metadata needed to find specific earth science data in a terrabyte sized archive. In addition, the object database must record knowledge on processing techniques in order to transform raw data into the desired end data products. Such a system would typically require significant amounts of storage in its own right, and must be able to support hundreds of simultaneous users. Considerable work in this area is being done by NASA Code 930.1 at GSFC in the context of RDC system software. We are exploring ways to utilize parallel I/O to improve throughput in database operations.

NASA

Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle High Performance Input/Output for Parallel Computer Systems		5. Report Date 7/1/96 6. Performing Organization Code
7. Author(s) W. B. Ligon		8. Performing Organization Report No.
9. Performing Organization Name and Address Clemon University Sponsored Programs Accounting and Administration Box 34535 Clemson , SC 29634-5355		10. Work Unit No. 11. Contract or Grant No. NAS5-32337 USRA subcontract No. 5555-20
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, DC 20546-0001 NASA Goddard Space Flight Center Greenbelt, MD 20771		13. Type of Report and Period Covered Final August 1993 -July 1996 14. Sponsoring Agency Code

15. Supplementary Notes

This work was performed under a subcontract issued by
Universities Space Research Association
10227 Wincopin Circle, Suite 212
Columbia, MD 21044

Task 20

16. Abstract

The goal of our project is to study the I/O characteristics of parallel applications used in Earth Science data processing systems such as RDC or EOSDIS. The approach is to study the runtime behavior of typical programs and effect of key parameters of the I/O subsystem both under simulation and with direct experimentation on parallel systems.

The key characteristic of this project has been studying the I/O behavior of real-life applications. The applications used in this study were taken from different levels of the typical RDC/DAAC processing cycle and include both image processing routines nd database search routines. In each case the applications were taken and run in their entirety. In several cases the codes were partial rewritten to improve their I/O characteristics. This approach was taken because in many cases the original programmers had not considered I/O performance but were focusing on the program's functionality. Each application was studied using both parallel and sequential I/O systems.

17. Key Words (Suggested by Author(s)) parallel computer systems	18. Distribution Statement Unclassified--Unlimited

19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of Pages 1	22. Price

NASA Form 1626 Oct 86