The Scalable I/O Initiative

B. Bershad, A. Chien, A. Choudhary, T. Cormen, E. DeBenedictis, D. DeWitt,
D. Ecklund, W. Gropp, R. Grossman, R. Kendall, K. Kennedy, C. Koelbel,
D. Kotz, K. Li, P. Lyster, D. Marinescu, P. Messina, R. Moore, S. O'Malley,
D. Payne, T. Pratt, D. Reed, J. Saltz, R. Stevens, S. Wallach, R. Williams *

February 23, 1993

1 Introduction

This white paper describes a collaborative project that brings together systems software developers, computer vendors, and applications teams to develop hardware and software systems to support scalable I/O for high performance computer systems. The project is organized around the provision of a full-scale testbed for the development and evaluation of new systems software for scalable I/O. In addition, research projects will be formed to address the scalable I/O problem from a number of perspectives, such as languages, compilers, file systems, networking software, persistent object stores, and low level system services.

Unlike the current state of parallel operating systems where a commonly available software platform exists (i.e., Mach), vendors that wish today to provide parallel I/O capabilities for their MPP systems must largely work from scratch when developing the file systems and user software needed to support a scalable I/O system. This problem forces many vendors to duplicate effort, and leads to the provision of conflicting solutions to the end users. This project intends to provide a commonly available set of software modules that can serve as the foundation for the next generation of parallel I/O systems.

To address this problem, we believe that systems software developers must work closely with both application developers and system providers to address the important issues. We also believe that it is critical for a full-scale I/O testbed to be available for development, full-scale debugging, and full-scale performance analysis and testing with real applications.

The I/O problem can be subdivided into a number of critical elements. First is the architecture of the I/O systems themselves. In this project we make few assumptions about the physical configuration of the system and I/O devices other than that they will be scalable with respect to the memory and CPU performance of the system. Next is the support for the devices themselves, provided by or in conjunction with the native operating system. We will develop as part of this effort parallel file systems that work with the native OS. Layered onto the parallel file systems are interfaces that provide access to the parallel I/O system, both for compilers and user code. A special component of this effort is the investigation of paradigms that can be embedded into High Performance Fortran (HPF) to support parallel I/O. Still higher-level functions will be provided by persistent object storage managers. Checkpoint and memory server capabilities may also be

*See Appendix A for Author Institutional Affiliation.

layered onto the parallel file systems or, in some cases, directly onto the storage devices. Since no parallel supercomputer is likely to be completely self contained, support for external file systems and remote devices via high-speed networks is also an important part of this effort. In particular, mass-storage systems (MSS) are likely to be connected via networks; accessing files that are stored in the MSS must not be a bottleneck. Applications projects will be sought out at each stage of development and testing to ensure the systems address real user needs. Finally, since the effort is organized around a testbed machine, full-scale experimentation can be used to refine the concepts and implementations.

Our goal is to develop and demonstrate an integrated set of tools and software systems that will provide to the user a scalable I/O facility. This software will be made available to the US high-performance computing community for productization and for future development.

2 Applications Motivation

Many "Grand Challenge" applications suffer substantial overhead due to the inadequate I/O speeds of current parallel computers. When these applications have access to teraFLOPS machines with tera byte memories, the I/O needs will increase dramatically. In some cases, programs that are not I/O-bound on 10 gigaFLOPS machines will become severely I/O-limited on the teraFLOPS computers that will be built within a few years unless I/O capabilities increase. A few examples of the I/O requirements of applications from the CSCC user community illustrate the magnitude of the problem.

Visualization of Planetary Data MPP systems are currently being used to assist in the understanding of planetary data obtained from NASA planetary space probes. In particular, terrain rendering is used to create three-dimensional perspective views of planetary imagery data, allowing a scientist to study the planet's surface as though he were close to it. Real-time image rendering of data sets from the Magellan mission which involve 200 MB of data per input frame require an I/O throughput of 13 GByte/s. The rendering can be done four frames a second even on today's relatively slow machines like the Intel Touchstone Delta. This requires 800 MByte/s just for the input stream, which is far in excess of what is currently possible.

Modeling Materials-Processing Plasmas Plasma-assisted materials processing is a key technology in many industries, particularly in microelectronics fabrication. Studies are currently underway on MPP systems directed at the calculation of accurate low-energy electron-molecule collision cross-sections from quantum-mechanical first principles, with an emphasis on small polyatomic molecules relevant to plasma processing. Promising areas of research which will be undertaken over the next several years would require an I/O rate approaching 400 MByte/s for a one-electron basis set $\sim$ 100 and the number of coupled scattering channels $\sim$ 40 on a system such as the Intel Touchstone Delta. The ratio of the I/O time to the compute time scales linearly with the number of coupled scattering channels, and important applications that are currently proposed to be undertaken will require $\sim$ 100 coupled channels. I/O rates of several GByte/s will be necessary to support these proposed studies.

Global Climate Modeling and Data Assimilation Present day General Circulation Models (GCMs) of the atmosphere involve on the order of 10^6 state variables. Observational data are assimilated into these models and used to force the simulation process to converge to a realistic

solution. In principle, if a complete set of error covariances is kept in the assimilation analysis, the number of variables is squared (i.e., it becomes 10^{12}). In the future, both the number of state variables, and the amount of observational data are expected to increase. Of order 10^{12} state variables is a goal. This becomes the size of files that need to be checkpointed as well as passed between modules. NASA's Earth Observing System (EOS) in the coming decade will focus on global earth observational datasets. Through the EOS DATA and Information System (EOSDIS) it is expected that up to a terabyte of data per day will be available for data assimilation. Long term climate studies can involve the simulation of the global atmosphere for 100's to 1000's of years.

3 I/O Characterization and Instrumentation

(Choudhary, Kendall, Pratt, Reed, Saltz)

Depending on the application, I/O can involve the transfer of temporary or persistent data within the system, between the system and network-connected storage devices, or between the system and specialized I/O devices. Based on our current understanding of user needs and requirements, we recognize three types of I/O:
(1) I/O for (possible) re-use of data. This involves the movement of data within the system. Examples are communication of data between processors, the writing/reading of checkpoint data, temporary storage of data during a computation (out-of-core solvers), and "permanent" on-line storage of data.
(2) Network I/O. This involves the movement of data into or out of the system over networks. Examples are file transfers between network-connected systems, data transfer associated with distributed file systems (such as AFS or NFS systems), data transfer to network-connected mass storage devices, and data transfer for distributed computing.
(3) Streaming I/O. This involves the transfer of data to or from "streaming" devices. Examples are data input from high data-rate instruments (sensors), data transfer to certain tape drives, data transfer for instrument control, and output to graphics devices (e.g., frame buffers).

The studies undertaken in this section will help refine this understanding.

Given the input/output requirements of the Grand Challenge problems we believe the first step in developing appropriate designs for scalable parallel input/output systems is a detailed characterization of the input/output behavior of large application codes on conventional supercomputers and massively parallel systems, particularly codes that are input/output bound for all or part of their execution (i.e., codes that severely stress the input/output subsystem). The input/output requirements of such applications determine the overall requirements for input/output subsystem performance, data storage and transmission volumes, acceptable system configurations, and software for access to data on secondary and tertiary storage.

The second step is to understand how current input/output systems respond to application input/output demands by capturing not only dynamic traces of application program input/output requests but also the dynamic pattern of interaction between operating system, file system, and input/output library components. To understand the these interactions, and to provide a springboard for parallel input/output research, we propose to instrument the OSF/1 code on the Paragon to capture detailed, dynamic performance data, with emphasis on the parallel input/output subsystem.

The Intel Paragon XP/S system, with a Mach microkernel and OSF/1 application interface on individual nodes, virtual memory, parallel disk arrays, and high-speed communication links, is a rich, unexplored software environment. Although small, inexpensive, high-capacity disks are now

widely available, and commercial, massively parallel systems support multiple, redundant arrays of inexpensive disks (RAIDs), little is known about high-performance parallel input/output in a multiuser, scientific computing environment. More tellingly, it seems unlikely that simple extensions of sequential file systems are appropriate for scalable parallel systems. Although the Intel Paragon system is our intended test platform, the results of this analysis are not limited to Intel hardware, but will be directly applicable to scalable parallel systems from a wide variety of vendors.

The third step is to build these understandings into a set of templates, benchmarks, evaluation methods, abstract models, and evaluation tools that allow realistic evaluation of alternative input/output subsystem designs for the Paragon (and, by extension, of other massively parallel input/output subsystems).

Specifically, we propose four basic research thrusts: (1) instrument and analyze the input/output behavior of a set of realistic application codes, (2) instrument the OSF/1 operating system source code on the Intel Paragon system to study and understand input/output system software dynamics, (3) use the application and operating system data obtained from instrumentation to develop a set of input/output templates, abstract models and benchmarks, and (4) use the benchmarks as a source of input/output subsystem activity that is scalable and repeatable, and, by execution of the benchmarks on simulators or actual systems, evaluate the capabilities of alternative input/output subsystem designs.

Our previous and current work in performance instrumentation and input/output data analysis provide the basis for our belief that analysis of detailed performance data is the key to understanding the dimensions of the input/output problem. In particular, we have (1) studied the performance of Intel's previous generation parallel file system, (2) examined the pattern of accesses to file archives at national supercomputer centers, and (3) are now studying application input/output access patterns at NASA. The common result of all these studies has been the high variation in resource demands and performance sensitivity to file access patterns. Understanding and exploiting this sensitivity is only possible if one can capture and analyze the interaction of operating system software components that support input/output and their response to application input/output request patterns.

In short, broad-based, experimental data capture and analysis are the prerequisites for a systematic attack on the scalable input/output problem. Only by analyzing existing applications and systems can we identify the important performance limiting factors. Moreover, the performance data and instrumented software provide a basis for performance comparisons and for further experiments.

Our research goals are driven by the need to capture detailed performance data from the application and system software levels. This, in turn, motivates the development of the requisite performance tools. Our research goals are: First, extend the existing Pablo performance instrumentation and data analysis tools to capture application input/output access patterns and related performance data. Second, instrument the Paragon XP/S OSF/1 operating system to capture data on the dynamics of current resource management algorithms, with emphasis on those algorithms that affect input/output performance. Third, measure the input/output behavior of large application codes on both current vector machines and, using the newly developed application and operating system instrumentation, on scalable parallel systems. Fourth, based on the performance data analysis, build a realistic set of application input/output behavior templates that embody typical patterns of computation and input/output and use the templates to construct a set of well-understood synthetic input/output benchmarks. Finally, "close the loop" by studying the effects of application and system software changes by other members of the consortium.

4 Compilers, Runtime System and Languages

(Choudhary, Kennedy, Koelbel, Saltz)

In order to achieve good scalability in I/O performance at the application level, appropriate support from compilers and runtime systems is needed. Furthermore, language extensions and directives are required that allow I/O parallelism to be explicitly specified, thereby enabling the compiler and runtime software to further enhance the performance of the I/O systems.

4.1 Compiler Support

In this project a key source of input and output in Fortran programs will be attacked. Even on massively parallel machines, there is a need to deal with data structures, particularly arrays, that are too large to fit in the system memory. Such data structures are generally referred to as "out-of-core" arrays. A significant portion of programmer time can be devoted to managing out-of-core arrays. Further, on parallel machines the problem is compounded by the need to take advantage of parallel input and output to achieve acceptable performance.

In Fortran D and High Performance Fortran, it is possible to specify the distribution of arrays across the processors of the parallel system. The language compiler then generates the code to send and receive off-processor data when it is required. We propose to extend the Fortran D/HPF mechanism to out-of-core arrays. The user would declare an out-of-core array in the same way as a normal array, except for adding the keyword "out-of-core" to the declaration. The programmer would specify distributions for out-of-core arrays, in the same manner as normal arrays, but the compiler would now be responsible for generating the input and output needed to move data to the point of usage along with any required communication. The user would simply use an out-of-core array like any other array.

In addition, various compiler optimizations are necessary for enhancing I/O performance in parallel programs, e.g., for recognition and parallelization of I/O operations. This will involve developing compiler techniques and transformations to schedule and perform I/O operations in parallel for various types of files and data formats. Specifically, I/O operations for HPF will be investigated. Compiler optimizations are necessary to support the overlapping of computation of one stage of the program with the I/O of another. Finally, compiler optimizations are necessary to support the efficient mapping of data between the I/O systems and the data distribution on the compute nodes. If this mapping can be determined at compile time, this information can help generate efficient schedules for parallel I/O. The same information can be used to generate schedules for efficient checkpointing.

4.2 Runtime Support

Runtime support for scalable I/O will be studied in several areas. One area of research is the development of methods for anticipating I/O access patterns in an effort to reduce the performance impact of I/O latencies. The approach will be to develop algorithms and software that use runtime information to prefetch data from the I/O system. Also, runtime information will be employed to partition arrays dynamically.

Another area of research interest is the development of runtime support and primitives to perform efficient collective I/O. We will develop techniques that use system parameters such as number of I/O nodes, stripe size, data distribution on the compute nodes and I/O nodes, number of compute processors, etc., to determine a good strategy for performing I/O, and then dynamically

redistribute data as required by the data decomposition on the compute nodes. This will also require maintaining a map between data distribution on I/O nodes and compute nodes.

We also propose to employ runtime characterizations of data access patterns to prefetch data to hide latencies, map data to I/O devices to reduce latencies, and reorganize I/O requests so that each processor can issue particularly advantageous patterns of I/O requests. Reordering I/O requests should be able to reduce I/O latencies by increasing the amount of disk data that can be cached on processor, and reducing the disk seek time by rearranging I/O requests to take advantage of knowledge about how data is stored on disk.

4.3 Language Support

It will be necessary to provide extensions to and directives for current programming languages to support parallel I/O. This will be accomplished by enabling a user to explicitly specify the parallelism associated with data structures in the program and their distribution across the parallel system, files accessed by the application and their distribution on the I/O devices, and the access methods for controlling the movement of data between the two. Explicit specification of I/O parallelism will enable compilers and runtime system software to further enhance the performance of the I/O systems.

We propose to develop directives that provide information about data distribution and alignment within application programs. Additionally, we propose to study language extensions, especially for HPF, to specify parallel I/O operations. Primitives will be developed that can be used from "node + message-passing" programs and from parallel languages, like HPF, to perform parallel I/O operations.

5 Scalable Persistent Object Stores

Providing scalable I/O from tera and peta byte secondary and tertiary storage to teraflops of computing is part of the solution to grand challenge problems. The scalable I/O solution includes new paradigms of referencing and naming data (e.g. metadata), incorporating these reference models into file-access systems that provides data access and data archiving, and finally delivery of the data to the application via the fastest and user productive means possible. This is part of a system architecture that includes 1000's of processors, 100's of gigabytes of physical memory, and an external hierarchical non-volatile disk and tape subsystems.

A major task of this initiative is to design, develop and implement a balanced, scalable, and modular persistent object store to assist in the scientific computing tasks of the Scalable I/O Laboratory.

5.1 Tightly-Coupled Stores

We propose to pursue a radically different approach for providing scalable I/O to scientific and engineering computations. As the underlying hardware architecture, we assume that an important component of the mass storage hierarchy will consist of disks that are tightly coupled with each processor rather than being a segregated component of the system. This is the approach used by all parallel database systems including Teradata whose largest configurations contain over 300 processors and 600 disk drives. This consensus architecture scales indefinitely. Furthermore, over the next five years as as standard disk drives shrink from a 3.5 inches to 1 inch, it will become possible to mount several disk drives directly on the processor cards. A suitable system for implementing and experimenting with this approach will be created as part of this scalable I/O initiative.

Given this basic architecture, it is straightforward to construct a parallel-file system that distributes the blocks of the file system across an appropriate number of disk drives; small files across a subset of the disk, large files, or those for which maximum I/O bandwidth is required, across all drives. A prototype of such a file system for a 64 processor/64 disk Paragon is already under development at the University of Wisconsin.

While this approach will indeed provide a scalable file system, we propose a much more radical approach. It is our contention that scientific applications should NOT deal with the file system in terms of physical blocks of data (e.g. read the next 512 bytes). As an example, consider a very large 2D array. Today the programmer is responsible for mapping pieces of the array to the file system. This is tedious and very inflexible. If the application is moved to a different parallel processor with a different I/O architecture, large pieces of the application may need to be rewritten. Instead, we feel that applications should deal with mass storage at a higher semantic level. In particular, we propose to replace the conventional file system with a parallel persistent object store. There are a number of reasons why we advocate such an approach:

(1) The languages for such systems allow programmers to manipulate both transient and persistent data (e.g. a matrix or a complex data structure) in the same way, freeing the programmer from having to do explicit file I/O. The CAD/CAM community has already begun to realize significant gains in programmer productivity through the use of such persistent programming languages. This community has also demanding performance requirements and they have found that this increase in productivity does not imply a corresponding loss in performance. The High Performance Fortran language has features that should mesh well with this approach.

(2) Too much important information is lost when data is stored "Unix-like" byte-stream files. A scalable I/O system based on the concept of typed persistent objects will provide a higher level semantics for the data as the application programmer will be provided a full type system for describing his/her persistent data. In addition, since the type descriptor for each persistent object is itself a persistent object, the meta information about the actual data will never be accidentally displaced.

(3) Very large objects can be transparently partitioned across multiple disk drives. As an example, assume the existence of a 2D persistent matrix class. Associated with the class will be a set of predefined methods for storing and manipulating instances of the class. As an example consider the create method (i.e. new() in C++). As parameters, this method might accept both the number of disks used to store the object and the declustering strategy for mapping the elements of the matrix to mass storage. For example, a "row-wise" declarative would distributed the elements of each row across the specified number of drives.

In such an approach, the actual computation (e.g., an eigensolver) would itself be a method accessing the matrix as if it were memory resident (regardless of its size). By suitably redefining the subscripting operator (i.e. [ ]) to incorporate the declustering factor and strategy, access to elements of the matrix can occur transparently, regardless of what disk actually holds the desired data. Such an approach avoids having the declustering factor or strategy from being "wired" into the application, allowing the user to tune the application by modifying each without having to rewrite the application itself (only the data will have to be reorganized). We plan on developing a full set of such classes for use in numerical applications.

5.2 Network-Connected Stores

We also intend to examine whether the strategies described above can be extended to handle objects on tertiary storage by having the first reference to an object cause the object to be staged from

tertiary to secondary storage.

As the basis for this system we will use the *Shore* persistent object manager that is currently under development at the University of Wisconsin. *Shore* has been designed to provide type persistent objects in both client-server and parallel computing environments. *Shore*'s type system is based on ODL, an industry standard for defining persistent objects.

Additional approaches that will be pursued include a modular storage system for controlling the persistent store that adheres to standard interfaces. Specifically, we would implement storage system software that is designed in compliance with the IEEE Storage System Reference Model. This would ease and enable the transition to new physical media and storage devices within the storage system.

We will also investigate developing a reference model for a persistent object store to encourage alternate implementations of essential components.

6 System Services

We propose three major thrusts in system services. The first involves development of checkpointing and memory hierarchy management techniques in scalable parallel I/O framework. The second involves the development of the fundamental I/O system software – interface, algorithms, and implementation to support both compiler controlled and explicit MIMD I/O. The third involves networking, an essential element of I/O with the advent of networked secondary and tertiary store.

6.1 Checkpointing and Memory Hierarchy Management

(Li)

Checkpointing and memory hierarchy management such as memory servers, shared virtual memory and persistent shared virtual memory are necessary support for grand challenge applications and will enhance the usability of massively-parallel machines. Concurrent checkpointing and logging utilities allow long running jobs to save state from time to time so that programs can be restarted from their saved states on stable storage, tolerating hardware and software errors, network disconnect situations, dedicated system time interruptions and unexpected system crashes. Checkpointing performance is particularly important in grand challenge applications because they have enormous memory images and long running times. Important challenges include developing message synchronization techniques to provide a consistent snapshot of the machine state, and concurrent checkpointing techniques that effectively use parallel I/O resources, to achieve high performance and eventually, real-time checkpointing and restart.

Memory hierarchy management techniques for massively parallel multicomputers include memory servers, shared virtual memory and persistent shared virtual memory. Research in memory servers is the first step towards utilizing the entire physical memory in a multicomputer. The memory server model extends the memory hierarchy of multicomputers by introducing a remote memory server layer between the local physical memory and fast stable storage such as disks. The memory server model takes advantage of both fast routing networks and available memory resources in multicomputers to reduce page transfer time by three orders of magnitude.

Shared virtual memory allows applications to exploit all of the memory capacity on existing multicomputer systems by providing a large coherent shared virtual memory space. Challenges in the memory hierarchy management include scalability, which is the current focus of designing and implementing memory servers and shared virtual memory systems for massively parallel multicomputers. The system data structures designed for these systems for small-scale multicomputes are

no longer appropriate. For this kind of research, it is essential to perform experiments at scale.

The persistent shared virtual memory research supports shared, persistent storage systems and is based upon shared virtual memory and parallel backing storage I/O. Persistent shared virtual memory keeps certain pages persistent across both program invocations and machine crashes, providing a highly concurrent persistent object base that allows its persistent mapping to take full advantage of concurrent I/O bandwidth, while supporting multiple threads of program execution. It is an integration of our previous research efforts in shared virtual memory, concurrent real-time checkpointing, concurrent pipelined logging, and concurrent real-time garbage collection.

6.2 Operating Systems and File Systems

(Chien, Reed, Cormen, Kotz, DeBenedictis)

The rapid increase in processing power available due to the advent of scalable parallel machines is reducing the ratio of input/output bandwidth to computing power, increasing the importance of input/output software to achieving high performance. Without major advances in software technology, the relative input/output performance of the new generation of scalable parallel systems will decrease tenfold, and a substantial fraction of all high-performance applications will become input/output limited.

We propose to investigate high-performance input/output software techniques for distributed memory, massively parallel systems, with emphasis on the Paragon XP/S system, that will achieve high-performance input/output through effective management of the parallel storage hierarchy and aggressive locality exploitation, based on observed, predicted, and algorithmically scheduled file access patterns. The design and performance of the software will be driven by previous and ongoing (see I/O Characterization and Instrumentation section) analyses of application input/output demands and a detailed performance instrumentation of the Paragon operating system software. The underlying hardware model is a secondary storage system with input/output nodes that each support one or more secondary or tertiary storage devices (i.e., disks, disk arrays, or file archives). Parallel applications that execute on compute nodes request file services by sending messages through the interconnection network to the input/output nodes.

This work will require development of new input/output libraries, modified file systems, and modifications to the existing OSF/1 operating system code on the Paragon XP/S system. The resulting software will be distributed as a series of releases to members of the Scalable Input/Output initiative and, through the Intel Supercomputer Systems Division, to its commercial customers. Thus, it will serve not only as a production file system, but also as a testbed for the design of future file systems. Though our studies use the Intel Paragon XP/S as an experimental vehicle, our objective is to resolve the key issues in scalable input/output for all massively parallel systems. We anticipate having access to other systems in the future on which we will validate the portability and efficiency of the software for diverse platforms.

The data management algorithms that form the heart of our input/output software will be based on the analysis of parallel input/output traces from both application programs and synthetic file access workloads; techniques for acquiring this data are the subject of research by other members of the consortium. An instrumentation of the OSF/1 operating system software for the Paragon XP/S will allow us to study the dynamics of existing file system components and understand the underlying reasons for observed performance, both with the extant OSF/1 file system software and our modifications. In both cases, the DARPA-sponsored Pablo performance analysis software will provide the requisite performance instrumentation and data reduction tools.

In high-performance computing systems, the input/output subsystem often represents a sub-

stantial fraction of the system cost. Consequently, determining how to balance configurations for particular workloads is an important practical problem. We will develop performance models which accurately characterize (predict) the obtainable input/output performance of a workload with respect to a particular input/output configuration.

The distribution of data across multiple disks increases vulnerability to disk or input/output processor failures. Since redundancy techniques such as RAID are designed for uniprocessor machines, new techniques are required for multiprocessors with distributed disks. We will develop methods to provide file availability even in the face of disk or input/output-node failure to determine the best methods for the multiprocessor environment, and the impact of these methods on performance.

Implementing file data management algorithms via libraries and user-level file servers is attractive because it allows the parallel input/output system to be developed atop the existing OSF/1 software. When possible, working above the operating system interface significantly reduces the complexity of experimentation. In addition, such a separation insulates the parallel input/output software from the underlying operating system software and input/output devices, increasing the portability both over machine configurations and software releases and across massively parallel machines from different vendors.

6.3 Networking for Efficient Scalable I/O

(Bershad, O'Malley)

The networking/communications architecture of a host has two distinct impacts on the search for a solution to the scalable I/O problem. First, networking teraflop machines together is a scalable I/O problem of the first order. Second, all I/O performance will be critically affected by the performance of the communications architecture. Many I/O devices will only exist on a subset of the nodes of the machine and hence I/O operations to those devices will require internode communication. Work is needed in both intra-host and intra-node network, if the performance goals of the scalable I/O initiative are to be met. More importantly the work in both areas is inter-related and must be coordinated on a system wide basis.

Layer Integration Recent work in the design of efficient network software has demonstrated the criticality of integrating of program communications strategies across a variety of system layers. Layer integration achieves two fundamental goals: efficiency and transparency. Layer integration allows strategies to be coordinated across levels. For example, failing to integrate buffering strategies across layers can require copying every time data crosses a layer boundary, seriously degrading system performance. Transparency without efficiency is irrelevant in high performance systems – transparent mechanisms which give inferior performance have historically been bypassed in high performance applications. Thus, each level of an integrated communications subsystem must perform as well as the best non-integrated mechanism.

Layer integration leads to a variety of optimizations which are critical to the I/O performance of teraflop class machines. Some combination of optimizations such as application level framing (allowing the application to specify the smallest unit of data dealt with independently reducing sequencing constraints), integrated buffer management (using the the same buffer management abstractions across all layers of the communication subsystem reducing copying), and layer bypass (removing unnecessary intermediate layers). Note that in some sense layer bypass and integrated buffer management are competing optimizations which both have the same goal.

Striping The purpose of striping is to increase I/O performance by spreading a single logical I/O operation over a parallel collection of I/O devices. The critical feature of network striping is that it allows a single application to transparently use multiple physical network links to achieve very high bandwidths. Such striping can multiply the communication rates achievable through parallelism and may even have cost advantages. At a minimum, implementing network striping requires new network protocols to drive multiple networks and new parallel communications which allow programmers to specify highly parallel network I/O operations in a network and machine architecture independent fashion.

Application level striping is an attempt to improve the performance of I/O in large scale multicomputers by integrating striping policies for the entire machine. A single system wide striping policy is used by all subsystems, and this policy is visible to the application writer. The goal is to perform each expensive operation (for example error detection/correction) once for the entire transaction and to avoid stalling the communications pipeline while waiting for out-of-order or dropped packets.

Support for scalable high speed I/O requires the careful integration of layers along the entire path from the application to the I/O devices. In addition, a system wide striping policy, integrated with the application level framing, is required. While the exact features of such an integrated policy are a critical area of research, any system which uses a collection of uncoordinated policies cannot hope to perform efficiently.

7 Scalable I/O Laboratory

Essential to the scalable I/O project is the development of prototype systems. The most important and unique resource is a large, easily reconfigurable system to look at large-scale effects and to run real applications. The system should be big enough to support Grand Challenge-like applications The Intel Paragon that the Concurrent Supercomputing Consortium is getting is an excellent candidate because of its fast internal communications speed and its modularity. In addition, it will be useful to have a few small, flexible, crashable systems for small-scale studies of hardware and software. Both types of system must run in a standard Unix environment. Both will also require periodic (two-year) upgrading as devices, channels, and processors improve.

To serve as a scalable I/O laboratory, the configuration of the large system will be augmented with several hundred node-connected disk drives, a large number of large memory nodes (128 MB instead of 32 MB), and 16 or more HIPPI interfaces. A network-connected mass storage system capable of supporting several HIPPI interfaces will be important for investigating the entire life-cycle of data.

It will be possible to carry out experiments with high-speed wide-area networks by using the CASA gigabit testbed project that will connect Caltech to Jet Propulsion Laboratory, San Diego Supercomputer Center, and Los Alamos National Laboratory. The link to JPL is already operational at 800 megabits/sec.

Appendix A: Author Institutional Affiliation

Brian Bershad, Carnegie Mellon University, bershad@ernst.mach.cs.cmu.edu
Andrew Chien, University of Illinois at Urbana-Champaign, achien@cs.uiuc.edu
Alok Choudhary, Syracuse University, choudhar@cat.syr.edu
Tom Cormen, Dartmouth College, thc@cs.dartmouth.edu
Eric DeBenedictis, Scalable Computing, 0005039494@mcimail.com
David DeWitt, University of Wisconsin, dewitt@cs.wisc.edu
Denise Ecklund, Intel Supercomputer Systems Division, denisee@ssd.intel.com
William Gropp, Argonne National Laboratory, gropp@mcs.anl.gov
Robert Grossman, University of Illinois at Chicago, grossman@eecs.uic.edu
Rick Kendall, Pacific Northwest Laboratory, d3e129@pnlg.pnl.gov
Ken Kennedy, Rice University, ken@rice.edu
Chuck Koelbel, Rice University, chk@rice.edu
David Kotz, Dartmouth College, dfk@wildcat.dartmouth.edu
Kai Li, Princeton University, li@cs.princeton.edu
Peter Lyster, Jet Propulsion Laboratory, lys@hyper-sun9.jpl.nasa.gov
Dan Marinescu, Purdue University, dcm@cs.purdue.edu
Paul Messina, California Institute of Technology, messina@ccsf.caltech.edu
Reagan Moore, San Diego Supercomputer Center, moore@sds.sdsc.edu
Sean O'Malley, University of Arizona, sean@cs.arizona.edu
David Payne, Intel Supercomputer Systems Division, payne@ssd.intel.com
Terry Pratt, National Aeronautics and Space Administration, pratt@cesdis1.gsfc.nasa.gov
Dan Reed, University of Illinois at Urbana-Champaign, reed@cs.uiuc.edu
Joel Saltz, University of Maryland, saltz@cs.umd.edu
Rick Stevens, Argonne National Laboratory, stevens@mcs.anl.gov
Steve Wallach, Convex Computer Corporation, wallach@concave.convex.com
Roy Williams, California Institute of Technology, roy@ccsf.caltech.edu