High-Performance I/O for Massively Parallel Computers

Problems and Prospects

Juan Miguel del Rosario and Alok N. Choudhary
Syracuse University

The effectiveness of teraflops parallel computers for solving many Grand Challenge and other large problems will be severely limited unless I/O support and performance are drastically improved.

Over the past two decades, advances in semiconductor and integrated circuit technology have fueled the drive toward faster, ever more efficient computational machines. Today, the most powerful supercomputers can perform computation at billions of floating-point operations per second (gigaflops). This represents a growth of two to three orders of magnitude over the past decade.

Much of this computational capacity is being harnessed to undertake large-scale mathematical modeling and simulation of various physical, chemical, and biological phenomena in connection with a broad range of theoretical and practical endeavors. For example, scientists are attaining unprecedented levels of clarity and detail in areas such as climate prediction and control, air and water pollution and quality management, lattice gauge theory, quantum chromodynamics, large-scale structure and galaxy formation, vision, and cognition. In engineering, computational techniques are being applied to the design and test of anticancer agents, anti-AIDS drugs, aircraft wingfoils, modern combustion engines, oil reservoir simulations, and the like. This increase in capability is intensifying the demand for even more powerful machines. Computational limits for the largest supercomputers are expected to exceed the teraflops barrier in the coming years.

I/O in Grand Challenge applications. Many supercomputer applications, like those mentioned above, are among a set of scientific and technical Grand Challenges, an annual list initiated about a decade ago by Nobel prize-winning physicist Kenneth Wilson.[1] Aside from being extremely complex and requiring significant amounts of processing time, these applications often deal with enormous quantities of data. Current near-term high-performance applications involve from 1 gigabyte to 4 terabytes of data per run.

Although the main memory regions of supercomputers are extremely large, some applications manipulate more data than these memories can hold. Such applications are appearing more frequently and have very high I/O requirements. For example, current archival sizes for a Grand Challenge group typically range from 500 Mbytes to 500 Gbytes of storage, with a peak of 10 Tbytes. Scientists

0018-9162/94/ $4.00 © 1994 IEEE

Table 1. I/O requirements for Grand Challenge applications.

Application	I/O Requirements	Storage
Environmental and Earth Sciences		
Eulerian air-quality modeling	Current 1 Gbyte/model, 100 Gbytes/application; projected 1 Tbyte/application.	Secondary
	10 Tbytes at 100 model runs/application.	Archival
4D data assimilation	100 Mbytes-1 Gbyte/run.	Secondary
	3-Tbyte database. Expected to increase by orders of magnitude with the Earth Observing System — 1 Tbyte/day.	Archival
Computational Physics		
Particle algorithms in cosmology and astrophysics	1-10 Gbytes/file; 10-100 files/run.	Secondary
	20-200 MBps.	I/O bandwidth
Radio synthesis imaging	1-10 Gbytes.	Secondary
	HiPPI bandwidths minimum.	I/O bandwidth
	1 Tbyte.	Archival
Computational Biology		
Computational quantum materials	150 Mbytes (time-dependent code) 3 Gbytes (Lanczos code).	Secondary
	40-100 MBps.	I/O bandwidth
Computational Fluid and Plasma Dynamics		
High-performance aircraft simulation	4 Gbytes of data/4 hrs.	Secondary
	40 Mbytes to 2 GBps disk, 50-100 MBps disk to 3-inch storage (comparable to HiPPI/Ultra).	I/O bandwidth
Computational fluid and combustion dynamics	1 Tbyte.	Archival
	0.5 GBps to disk, 45 MBps to disk for visualization.	I/O bandwidth

anticipate that by the time teraflops machines with terabytes of memory appear, these I/O requirements will increase dramatically, in some cases more than 100-fold (as with climate modeling), reaching 10 petabytes per Grand Challenge group.

But memory capacity is not the only consideration. Supercomputers are commonly interfaced with various peripheral devices (such as external disk storage systems, mass storage devices, visualization devices, video cameras, networks, and other supercomputers) for pre- and postprocessing of data, or simply for additional working storage.

In many cases, the speed of access to data can determine the rate at which the supercomputer can complete an assigned job. (Such jobs, for which I/O — not computation — is the bottleneck, are said to be *I/O bound*.) The need to access data via network-connected remote devices introduces significant delays over access to the internal I/O subsystem. The expansion of support for global computing paradigms amplifies the severity of this problem. Today, most high-performance applications involve I/O rates of 1 to 40 Mbytes per second for secondary storage and 0.5 to 6 MBps for archival storage. Application developers indicate that probably 1 GBps to secondary storage and 100 MBps to archival store will be required in the near future.[2] To better understand the need for such high data-transfer rates, we provide the examples that follow.

Imaging of planetary data. The spacecraft Magellan has been collecting data from the surface of Venus since September 15, 1990. Using radar to penetrate surrounding Venusian cloud cover and to scan the surface for structural information, Magellan has transmitted to earth in excess of 3 Tbytes of data. Producing a 3D surface rendering at 200 Mbytes of data per frame would require more than 13 Gbytes per second at 50 frames per second. This far exceeds the I/O capacity of today's machines. Rendering a portion of the Venusian surface on a 512-node Intel Touchstone Delta takes several days.[3]

Climate prediction. Research efforts in climate and global change, long-range weather prediction, and land-

surface processes are crucial to understanding geographic, oceanic, and atmospheric systems. The most complex of these are the general circulation models of the atmosphere and ocean that must be capable of simulating geophysical fluid dynamics on appropriate scales. Current atmosphere/ocean models have certain requirements on an Intel Touchstone Delta. For a 100-year atmosphere run with 300-square kilometers resolution and 0.2 simulated year per machine hour, the simulation takes three weeks runtime and generates 1,144 Gbytes of data at 38 Mbytes per simulation minute. For a 1,000-year coupled atmosphere-ocean run with a 150-square kilometer resolution, the atmospheric simulation takes about 30 weeks and the oceanic simulation 27 weeks. The process produces 40 Mbytes of data per simulation minute, or a total of 20 Tbytes of data for the entire simulation.[4]

Table 1 summarizes the I/O requirements for several Grand Challenge applications. The data is based on presentations by scientists at the Grand Challenge Applications and Software Technology Workshop in Pittsburgh in May 1993.[2]

High-performance distributed computing. Today, many scientists share a vision for the future of high-performance distributed computing (HPDC); they envision a nationwide heterogeneous distributed-computing environment in which information and data will be shared, processed, and stored in a seamless, globally oriented manner.

The term *metacomputing*, originally used in the 1980s, refers to the concept of having several machines work cooperatively on a single problem. The recent popularity of this computing paradigm stems from the fact that a supercomputer's execution rate for a given application is a function of how closely the problem domain maps to the computer's architecture. Metacomputing allows the assignment of each task in a problem to the machine that can execute it optimally.

Figure 1 shows how a high-performance computing infrastructure might appear, with computational centers composed of various combinations of vector computers, massively parallel computers, multiprocessors, high-resolution visualization systems, tens to hundreds of workstations, mass storage and archival systems, and so forth, connected by network links of varying distances and capacities. The distributed nature of this computational paradigm would place a high premium on the I/O capacities within and between processing centers.

The nature of I/O in MPPs

I/O requirement characterization. The parallel I/O problem can be viewed from several perspectives: languages, compilers, file and runtime systems, networking systems, operating systems services, storage systems, and so forth. Crockett[5] classifies parallel file organizations into a number of categories based on a global and internal view of the access pattern. Existing parallel file systems such as Intel's CFS and Ncube's file system provide support for some subset of these file organizations.

The use of parallel computers is becoming more sophisticated, so it is important to reexamine what we understand about the nature of the I/O requirements. In particular, the following concerns arise. Our understanding of I/O requirements for scientific purposes stems primarily from past experience with supercomputing applications or very basic (in terms of I/O) parallel applications. Parallel I/O on distributed-memory systems will vary greatly from supercomputer I/O because of the difference in underlying hardware. The basic model for current parallel I/O systems includes an I/O subsystem architecture that is dis-

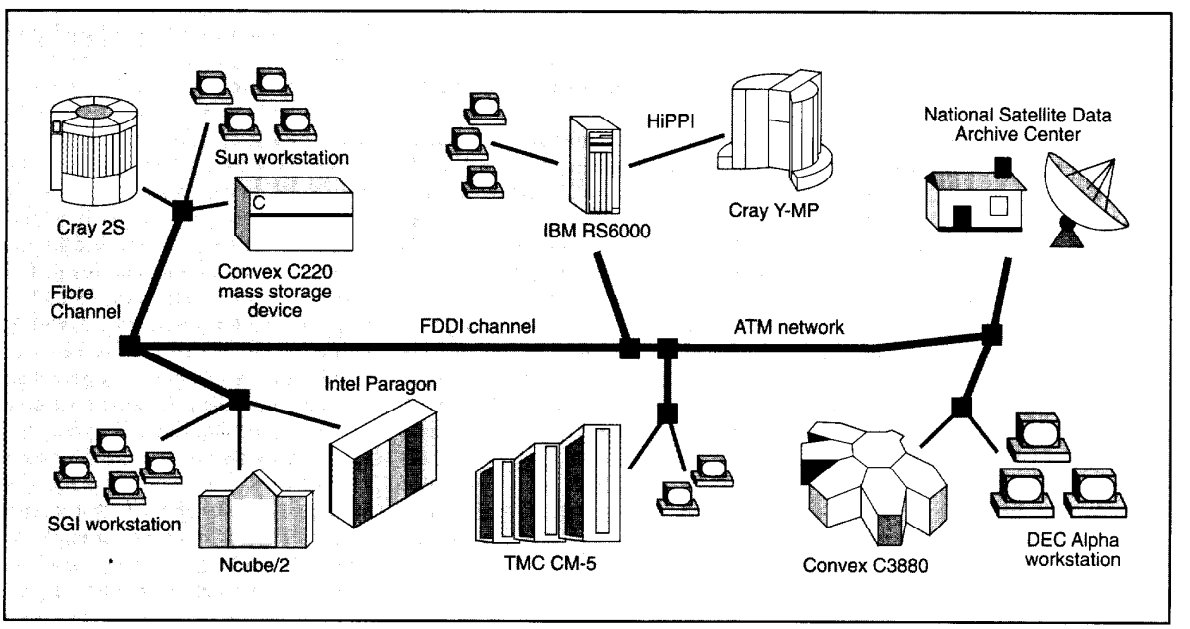

Figure 1. High-performance distributed computing network.

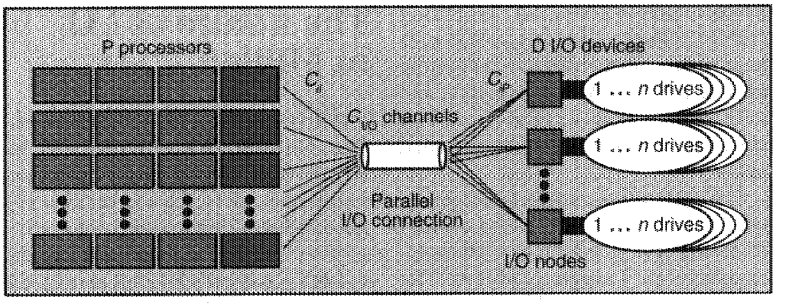

Figure 2. Parallel I/O subsystem architecture.

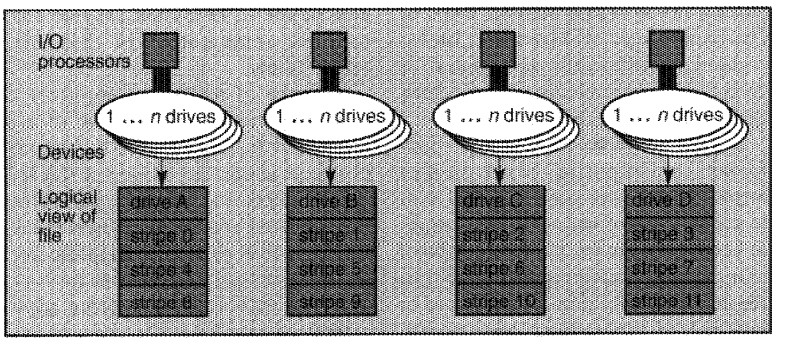

Figure 3. Parallel I/O device architecture.

tributed in nature and consists of independent I/O nodes that support one or more secondary storage devices. On the other hand, traditional supercomputers commonly have a large disk or disk arrays connected to the computational unit via a single bus line.

Another concern is that simply evaluating I/O-bound parallel applications might not provide a good representation of the actual I/O requirement, since the code was probably designed to optimize its performance on existing inadequate I/O systems and may be very tightly coupled to the particular machine architecture. As I/O subsystem architectures change, the lessons we learned from previous applications may no longer be valid. Finally, future fine-grained machines will change I/O requirements, perhaps making them more akin to those of transaction processing.

Therefore, we need to better understand the I/O requirement for parallel computing. In setting out to explore this area, we need to be aware of parameter dependencies and possible design bias in any benchmark application used.

Before the I/O problem can be properly addressed, quantitative assessments are needed for

(1) spatial and temporal data access patterns,
(2) current hardware and system software response to application I/O demands, and
(3) current performance bottlenecks.

Architectural model for distributed-memory I/O subsystems. The single overriding design objective in distributed-memory systems design is scalability. This property extends to the I/O subsystem and its communication channels. In considering the architectural model for distributed-memory I/O subsystems, we assume a fixed number of processors within the computational array and a fixed number of disks. Each disk is associated with an I/O node through a connected controller. Each I/O node acts as part of a distributed file server and operates as an intermediary between a set of disks and the computational array.

Figure 2 shows the system configuration. Each processor at the right side of the computational array has a direct connection to an I/O device. A specific example of this configuration can be found in the Intel Touchstone Delta architecture. This architecture is a 16×32 two-dimensional array of compute nodes with 16 I/O nodes on each array side; each I/O node has two associated disks.

The Intel Paragon, on the other hand, has a mesh topology with no special I/O channels. Separate I/O nodes are used but can be placed anywhere within the mesh network. Each I/O node has its own set of attached disks, as in our model; the set is maintained as a redundant array of inexpensive disks (RAID) level 3 device.

Although it follows the same basic model, the Thinking Machines CM-5 has a slightly different I/O device architecture. The CM-5 generally has fewer I/O processors, with a high-bandwidth channel connected to either a standard interface or to the CM-5 secondary storage device. This storage device, called the DataVault, is also a RAID level 3 device.

An I/O node, with an accompanying set of disks, can be viewed as a separate functional unit within the parallel I/O system. The organization between the disks and its I/O node can take on any of the numerous existing host-to-disk architectures. These range from the more conventional host/disk head-of-string type of format to any RAID-level architecture.[6] Figure 3 shows a typical I/O subunit comprising four I/O nodes and their associated disk drives.

The performance of each I/O device is measured by its ability to handle the workload from the computational array. As a first approximation, the workload applied on the set of disks in the parallel computer might be determined by the type of data distribution selected at the application level. For instance, cyclic distributions tend to require smaller transfers from disk, while block distributions tend to require the movement of large blocks of contiguous data. However, caching and/or prefetching within the computational array and/or on the I/O nodes contributes toward redefining the workload actually applied (that is, visible) to the set of disks. For example, an I/O node might group many small requests into a single large data request to its set of disks, resulting in better performance.

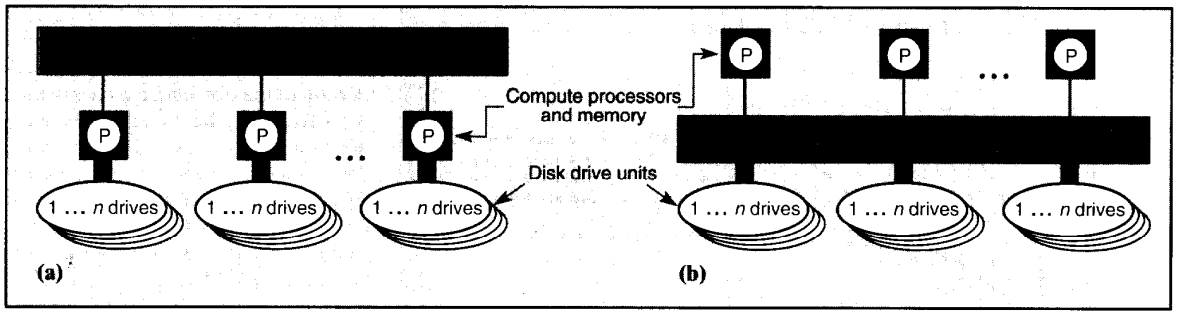

Figure 4. Tightly coupled secondary storage (a) versus loosely coupled secondary storage (b) compute and I/O systems.

Parallel files are distributed among the set of disks in the I/O subsystem by declustering the data across the disk array (a technique known as striping), as shown in Figure 3. Load-balance issues arise from the degree of correspondence between the application-defined data decomposition and the data-storage mapping defined by the stripe size. We discuss the relationship between stripe size and load balance in the next section.

Another major concern in parallel I/O architectures involves the data transfer bandwidth to and from I/O devices. This bandwidth is limited by the size and number of communication channels between the computational array and the I/O devices. In general, we can view the interconnection model as illustrated in Figure 2. The number of I/O channels $C_{I/O}$ between the computational array and the I/O devices is given by

$$C_{I/O} = \sum_{i=1}^{D} C_{iP} = \sum_{i=1}^{P} C_{iI}$$

where D is the number of I/O devices, and C_{iP} represents the number of connections from the ith disk to the computational array. P is the number of computational processors, and C_{iI} represents the number of connections (1 or 0) from the ith processor to an I/O device. The greater $C_{I/O}$ is, the greater the subsystem's data transfer capacity.

In an extension of the tightly coupled storage device model to general-purpose parallel machines, the disk units would be integral to the computational array. As shown in Figure 4, each disk unit is closely coupled to a processing unit, resulting in very low transfer times between the processor and its local disks and in higher overall I/O bandwidths. Reddy et al. discuss the problem of embedding I/O nodes in parallel computers in detail.[7]

Before such models can be completely adopted, however, the following questions must be addressed:

(1) What are the effects on compute performance of the extra I/O processing required of processors within the computational array?
(2) What limits are imposed by additional memory requirements for I/O buffering?
(3) What are the effects of additional contention in the interconnection network arising from increased I/O traffic within the computational array?
(4) How will latency-reduction issues (arising from reduced possibilities for overlapping I/O with computation) be addressed? and
(5) How are external devices (for instance, tape silos and networks) to be connected?

Operating and file systems

Providing the necessary support for parallel I/O at the lower levels of system software requires investigating existing algorithms for file management in distributed-memory parallel environments. Benchmark studies conducted on existing file systems let us identify deficiencies that must be addressed in our attempts to construct better parallel file systems.

Communication latency. Communication latency to the I/O nodes contributes greatly to the poor performance of existing systems. High latency dominates the overall transfer time of a sequence of small- to moderate-size requests. Hence, requests for data must be made in large chunks if they are to be efficiently serviced. This efficiency is incurred at the expense of using an access mapping with a more natural correspondence to the computational decomposition.

Another concern pertains to object-oriented operating systems such as the Intel Paragon OSF/1. Here, the object structures impose additional processing overhead, thereby increasing communication latency.

A reduction in overall application execution time may sometimes be possible by overlapping computation with I/O, which an asynchronous message-passing protocol makes possible.

Data decomposition. In constructing parallel file systems, we are concerned with providing support for user applications. A common programming paradigm in scientific computing involves decomposing the problem domain. This decomposition is translated into a data-domain mapping over the computational array (see the next section). In conducting I/O, the application must be able to preserve some correspondence between this mapping and the mapping of data over storage devices (for example, disks). The file system must accomplish this in an efficient manner. Current parallel file systems have little support for data decomposition or control over stripe size. Recent benchmark results show that existing file systems are extremely sensitive to I/O access patterns and that performance varies greatly as a function of data decomposition.[8]

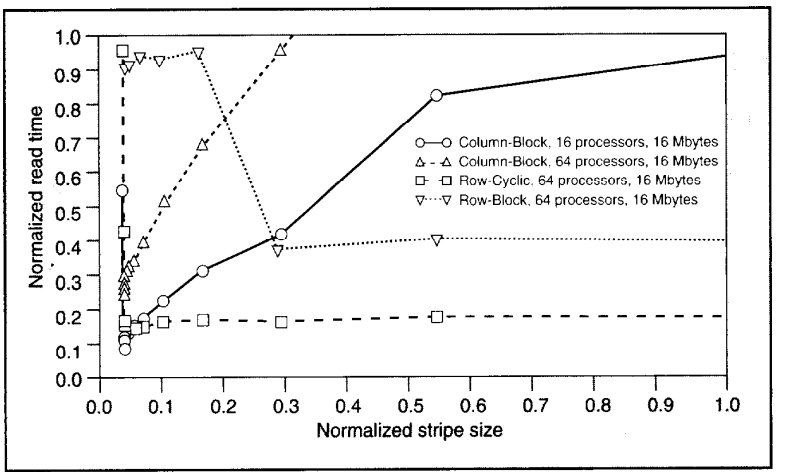

Figure 5. Read time as a stripe-size function.

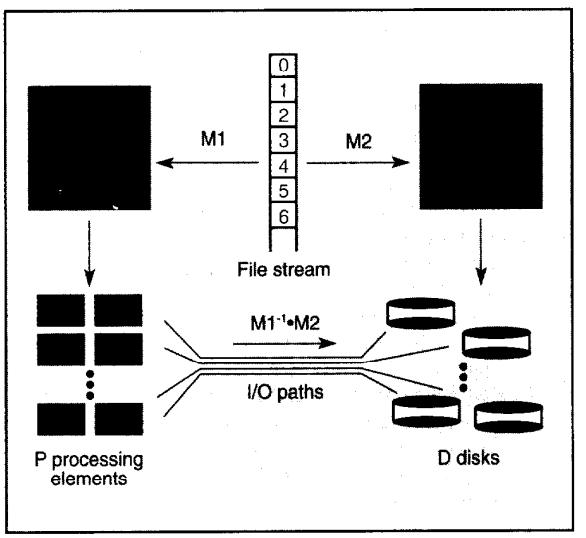

Figure 6. Parallel I/O mapping functions.

Furthermore, the load balance aspect of data access becomes critical. If the data decomposition selected by the application is incompatible with the stripe size, it is possible to overload a particular I/O node with requests, thus creating a severe bottleneck in the file system.

To illustrate load balance effects for various data decompositions, Figure 5 shows the time it takes to complete a read operation as a function of stripe size. The stripe size, which ranged from 64 bytes to 1 Mbyte in the actual experiment, is shown as a normalized value obtained by dividing each stripe size by 1 Mbyte. The read times associated with each curve are also normalized by dividing over the largest read time taken for that curve. The data was collected from an Ncube/2.[9]

Hence, the file system must be able to accommodate various data distributions and efficiently manage various permutations of data decomposition versus decluster mappings.

Data mapping in parallel I/O. In programming a parallel computer, data decomposition is often used as a method of obtaining some degree of parallelism that is usually easy to manage and typically matches the problem domain closely.

When I/O has to be performed, each compute node must have some knowledge of where the data belonging to its portion of the distributed data structure is located. In other words, a mapping function has to be established from the data structure element to the relevant disk block. To establish such a mapping from the processor array to the distributed file, two submappings need to be considered.[10] The first, M1, involves mapping data over the set of processing elements. File-data organization over the set of disks represents the second mapping, M2.

For parallel I/O to be efficient, both mappings must be resolved into a data transfer strategy, as shown in Figure 6. The current parallel file system on the Ncube-2 resolves these mappings internally into a single data-transfer mapping, which computes proper source and destination addresses during file-data access. The Intel Touchstone Delta file system (the CFS) maintains only the M2 mapping, making the user responsible for managing the M1 map. This is called through *direct access*. Problems arise from this approach in cases where the first and second mappings resolve into a data transfer mapping (representing an access strategy) that results in poor I/O performance. Such problematic mapping pairs are quite common.[9]

File system functionality and interface requirements. The file system design is a key ingredient in determining the effectiveness of the overall parallel I/O subsystem. Through its functionality and interface, the file system defines the set of I/O operations that will be available to runtime systems and compilers.

Interface. Cormen and Kotz surveyed existing commercial parallel file systems, evaluating them on the basis of a proposed set of required capabilities.[11] Their results are illustrated in Table 2, which also includes support for application-level specification of data decomposition (in the last column) as an additional criterion (considered "n/a" for SIMD or shared-memory machines).

Their formulation for the set of necessary capabilities is founded on what might be required in order to perform a collection of commonly used algorithms, such as sorting, permutations, matrix transpose, fast Fourier transform,

Table 2. Capabilities of existing commercial parallel file systems.

File System	Control Over Data Declustering or Stripe Factor	Ability to Query the Current Configuration	Ability to Access Disk Blocks Independently	Ability to Turn Caching On or Off	Ability to Turn Parity On or Off	Ability to Recognize and Support (Optimize for) Data Distribution
Intel CFS	Limited	Limited	Yes	No	n/a	No
Paragon PFS	Yes	Yes	Limited	Yes	Limited	No
Ncube (old)	Yes	Limited	Yes	No	n/a	No
Ncube (new)	Yes	Limited	Yes	No	n/a	Limited
KSR-1	No	?	Limited	No	Limited	n/a
MasPar	No	Yes	No	No	No	n/a
TMC DataVault	No	Yes	No	No	No	n/a
TMC SDA	No	Yes	No	No	No	n/a
IBM Vesta	Yes	Yes	Yes	No	n/a	Yes

matrix multiplication, and matrix factorization; further support is taken from the results of previous empirical studies. Table 2 shows that existing file systems still have limited functionality. Until now, members of the high-performance computing community have not been able to agree on a standard interface for parallel file systems, although this is a critical requirement from a software engineering standpoint. The key questions in interface design concern how much information and control are made available to the user. It could be that explicit control of lower level configuration parameters (for example, block placement, striping, and prefetching) must also be made available. These capabilities enable the user to explicitly adjust for good performance. Furthermore, the system's ability to use additional access information enables it to perform optimizations instead of relying on general-purpose algorithms.

Prefetching and caching. Prefetching and caching of data within the I/O subsystem involves extending the solution applied to primary memory. The idea is to exploit locality of access by temporarily saving (caching) blocks that contain recently used data, the expectation being that nearby data also will soon be accessed. Prefetching is a predictive extension to caching and is based on longer distance distributions of locality. The critical issue here is the nature of locality. In the traditional view, locality is established by a sequential view of data access. However, in a parallel subsystem, this view may no longer hold.

From the point of view of an I/O server node, requests may arrive such that prefetching only every other block results in improved performance. The number of messages required to read the entire file and each request's latency cost can be cut in half.

Checkpointing. Users commonly share portions of parallel machines for production runs. A problem arises when the system must be reset due to a crash in a program sharing the machine. In this case, programmers must provide regular state-saving routines that will let a run start from a point just prior to the interruption. Concurrent checkpointing lets long-running jobs automatically save state at regular intervals so that they may be restarted after interruptions without unduly retarding their progress. This provides for fault tolerance of hardware and software errors, network malfunction, and system interruptions.

Suppose we have a program that we wish to checkpoint in 100 seconds. For each Gbyte that needs to be checkpointed, we need a 10^9 bytes/10^2 = 10^7 Bps = 10 MBps I/O bandwidth. For 1 terabyte of data, this translates to a 10-GBps I/O bandwidth; to attain such a bandwidth would require the use of 100 100-MBps High Performance Parallel Interface (HiPPI) channels.

Runtime system and compilers

Compiler and runtime system support for parallel I/O will maximize the system's ability to exploit user-supplied or source-level information to optimize I/O performance. The additional information will enable the formulation of improved I/O access schedules, which will result in a more effective communication latency hiding strategy.

Compilers. Various compiler optimizations can enhance I/O performance in parallel programs. Recognizing and parallelizing I/O operations are key ingredients here. We need to develop compiler techniques that will allow I/O operations to be parallelized for various file types and data formats. Mechanisms that permit user expression of I/O data structures would facilitate programming and functional interpretation of instructions by the compiler. Analysis methods similar to those for automatic decomposition should be investigated to enable the compiler to reschedule operations, overlapping I/O with computation. Moreover, compile time information on the access pattern used by the application can be supplied to

Table 3. Direct access versus two-phase access (64 processors 10K*10K array, time in msec).

Distribution	Best Read Time 10K*10K	Redistribution Time 10K*10K	Total Read Time 10K*10K	Direct Read Time 10K*10K	Speedup 10K*10K
Column Block	11,395	—	11,395	11,395	1.00
Column Cyclic	11,395	2,478	13,873	63,400	4.57
Row Block	11,395	1,028	12,423	78,767	6.34
Row Cyclic	11,395	3,092	14,487	n/a	>248.50

the runtime system to help generate efficient access and checkpointing schedules.

Runtime. Runtime libraries afford a level of insulation from operating system and file system software, making them attractive as a development environment. Providing parallel I/O support at this level increases the chance of portability.[12] Furthermore, by incorporating a comprehensive interface to compilers, additional compile-time information can be harnessed in formulating a data movement strategy for the application.

Although language extensions help us to represent data distribution information in a way that closely matches the underlying computation, provisions for language support that will allow similar specifications to be made with I/O expressions have not been sufficiently addressed. As a result, it is difficult and sometimes impossible to perform such a parallel I/O mapping in a manner that yields optimal performance. Current work on runtime systems uses composite mapping techniques to improve parallel I/O performance.

Composite mapping strategies. Experimental results show that file-system performance can vary greatly as a selected data-distribution function. The bandwidth for any given parallel I/O configuration is highly dependent on the file size; stripe-size-dependent factors (for instance, load-balance and request size) cause widely divergent access times for most distributions.[9]

Based on these observations, alternative schemes for conducting parallel I/O have been devised. One approach, a two-phase access strategy, uses combinations of mappings that improve average performance and guarantee greater consistency over a broader spectrum of data distributions. The idea behind this strategy involves dividing the parallel I/O task into two separate phases.

In the first phase, the parallel data access is performed using a data distribution that conforms with distribution of data over the disks. That is, we introduce an intermediate mapping M2′ and access data with M2′ = M1. In phase 2, we redistribute the data at runtime to match the application's desired data distribution (that is, from M2′ to M2).

By using the two-phase redistribution strategy, the costs inherent in many I/O configurations are avoided. Selecting a single, "good" configuration effectively reduces the bottleneck activity — I/O to the parallel device. Furthermore, the redistribution phase improves performance because it can exploit the higher bandwidths made available by the higher degree of connectivity present within the computational array's interconnection network. This strategy effectively decouples the user-selected data-distribution mapping from the file mapping to the disks (that is, the declustering mapping); this results in performance that is much less dependent on user selected mappings.

Table 3 compares access rates between the direct-access and two-phase-access strategies obtained from runs on an Intel Touchstone Delta.[9] Here, the "Best Read Time" is the time it takes to read the desired data into the computational array using a distribution that conforms to the distribution of data on the disks. This is phase 1. "Redistribution Time," the time it takes to redistribute the data, is phase 2. "Total Read Time," the sum of the first and second columns, represents the complete two-phase access. "Direct Read Time" is the time it takes to perform direct access. The last column shows the "Speedup" gained from using two-phase access over the direct-access method.

Another composite mapping strategy can be applied to "out-of-core" type applications. The idea is to extend the two-phase access with an additional mapping function that will define the relationship between portions of the out-of-core file. This is illustrated in Figure 7.

Networking technology

When data has to be transferred out of a computing environment to an external device or another remotely connected supercomputer (for example, for pre- or postprocessing, visualization, and so forth), the network's capacity becomes a critical consideration. As with I/O devices, network capacity has lagged behind memory and processor technology.

In the past few years, a number of technologies have been developed to improve network interface and capacity. The HiPPI standard, for instance, includes a mapping to IEEE 802.2 for supporting common network protocols like the Transmission-Control Protocol/ Internet Protocol. Interfaces to alternative layers are also under development. For example, the Intelligent Peripheral Interface (IPI-3) provides command sets for disk and tape, and will allow support for striped disks and tape devices directly connected to HiPPI channels or LANs.

Another technology that IBM is developing in collaboration with Lawrence Livermore National Lab-

Table 4. Network technology.

Type	Bandwidth	Distance	Technology
Fibre Channel	100-1,000 Mbps	LAN, WAN	Fiber optics
HiPPI	800 Mbps or 1.6 Gbps	≤25 m	Copper cables (32 or 64 lines)
Serial-HiPPI	800 Mbps or 1.6 Gbps	≤10 Km	Fiber-optics channel
SCI	8 Gbps	LAN	Copper cables
Sonet/ATM	55-4.8 Gbps	LAN, WAN	Fiber optics
N-ISDN	64 Kbps, 1.5 Mbps	WAN	Copper cables
B-ISDN	≤622 Mbps	WAN	Copper cables

LAN — Up to several meters
WAN — Up to several kilometers

oratory is called Fibre Channel. Aside from the use of an optical (as opposed to copper) medium, Fibre Channel encapsulates a more extensive set of services than does HiPPI. Unlike HiPPI, it targets up to 4,096 switch connections for distances of up to several kilometers. Furthermore, Fibre Channel supports multiple connection types (such as datagram and virtual circuit) over various physical layers (for example, coaxial cables, fiber, lasers, and LEDs) at multiple data rates.

The Scalable Coherent Interface standard allows development of local area networks with speeds of up to 8 gigabits per second and is about 10 times as fast as Futurebus+. SCI provides bus services (read, write, lock, and so forth) by sending packets over a large number of point-to-point links. Table 4 summarizes existing technologies.

Over the years, LANs and WANs have developed along independent, sometimes divergent, lines. Asynchronous Transfer Mode technology could provide a driving force to integrate LAN and WAN technologies. Implemented as the underlying support layer for B-ISBN, ATM provides a common framework for public wide-area networks as it does for local area networks.

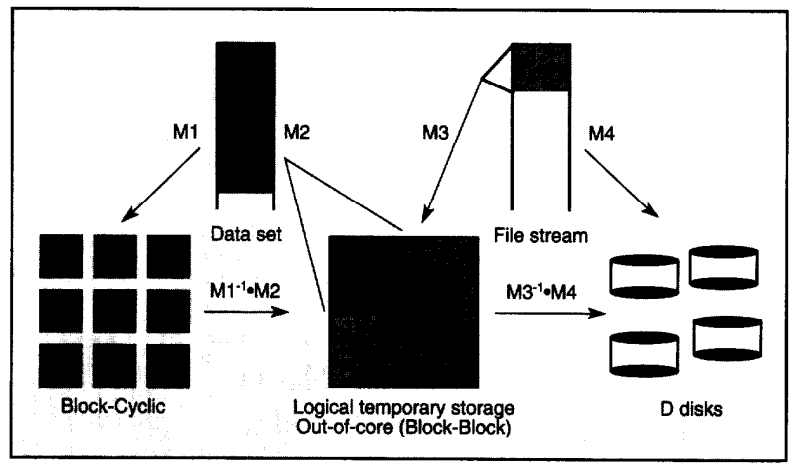

Figure 7. Composite mapping for parallel I/O.

The recurrent themes in the parallel I/O problem are the existence of a great variety in access patterns and the sensitivity of current I/O systems to these access patterns. An increase in the variability of access patterns is also expected, and single resource-management approaches will likely not suffice. Providing the I/O infrastructure that will support these requirements will necessitate research in operating systems (parallel file systems, runtime systems, and drivers), language interfaces to high-performance storage systems, high-speed networking, graphics and visualization systems, and new hardware technology for I/O and storage systems.

As a first step in this research, I/O access patterns must be quantitatively characterized by instrumenting multiple platforms and collecting trace data for large application codes. The knowledge gained from this step must be integrated into an evolutionary development cycle for I/O systems as a whole.

The area of parallel I/O is vast, with aspects related to many areas of computing in general. This article has only scratched the surface. Other relevant areas that need to be addressed include multimedia requirements that place different demands on the I/O system; database systems; parallel data transfer; fault-tolerance; distributed file systems (over an HPDC network); archival storage; and visualization. ■

Acknowledgments

This work was sponsored in part by ARPA under contract No. DABT63-91-C-0028. Alok Choudhary is also supported by the National Science Foundation Young Investigators Award (CCR-9357840). The content of this article does not necessarily reflect the US government's position or policy, and no official endorsement should be inferred. We also acknowledge the support

of the Center for Research in Parallel Computing and the Concurrent Supercomputing Consortium for providing access to their computing resources.

The following people aided our preparation of this article with ideas presented in lectures or through informative discussions: Brian Bershad, Rajesh Bordawekar, Andrew Chien, Tom Corman, David DeWitt, Denise Ecklund, Ian Foster, Geoffrey Fox, Garth Gibson, Bill Gropp, Ken Kennedy, Chuck Koelbel, David Kotz, Kai Li, Paul Messina, Regan Moore, David Patterson, David Payne, Larry Peterson, Terry Pratt, A.L.N. Reddy, Dan Reed, Joel Saltz, Marc Snir, and Rick Stevens.

References

1. "High-Performance Computing and Communications, Grand Challenges 1993 Report," *A Report by the Committee on Physical, Math., and Eng. Sciences Federal Coordinating Council for Science, Eng. and Technology*, Committee on Physical, Mathematical, and Engineering Sciences Federal Coordinating Council for Science, Engineering and Technology, Washington, D.C., 1993, pp. 41-64.

2. A. Brenner et al., "Survey of Principal Investigators of Grand Challenge Applications: A Summary," *Proc. Workshop Grand Challenge Applications and Software Technology,* Argonne Nat'l Labs, Chicago, 1993.

3. M. Grossman, "Modeling Reality," *IEEE Spectrum*, Sept. 1992, Vol. 29, No. 9, pp. 56-60.

4. I. Foster, M. Henderson, and R. Stevens, "Workshop Introduction," *Proc. Workshop Data Systems for Parallel Climate Models at Argonne National Laboratory*, Argonne Nat'l Labs, Chicago, 1991.

5. T.W. Crockett, "File Concepts for Parallel I/O," *Proc. Supercomputing 89*, IEEE CS Press, Los Alamitos, Calif., Order No. 2021, 1989, pp. 574-579.

6. D. Patterson, G. Gibson, and R. Katz, "A Case for Redundant Arrays of Inexpensive Disks," *Proc. SIGMod Conf.*, ACM Press, New York, 1988, pp. 109-116.

7. A.L.N. Reddy, P. Banerjee, and S.G. Abraham, "I/O Embedding in Hypercubes," *Proc. Int'l Conf. Parallel Processing*, IEEE CS Press, Los Alamitos, Calif., Order No. 889, 1988, pp. 331-338.

8. R. Bordawekar, J.M. del Rosario, and A. Choudhary, "An Experimental Performance Evaluation of the Touchstone Delta Concurrent File System," *Proc. Int'l Conf. Supercomputing*, ACM, New York, 1993, pp. 367-376.

9. R. Bordawekar, J.M. del Rosario, and A. Choudhary, "Improved Parallel I/O Via a Two-Phase Runtime Access Strategy," *Proc. Workshop Input/Output Parallel Computer Systems*, 1993, pp. 56-70.

10. E.P. DeBenedictis and Peter Madams, "Ncube's Parallel I/O with Unix Capability," *Sixth Distributed-Memory Computing Conf.*, IEEE CS Press, Los Alamitos, Calif., Order No. 2290, 1991, pp. 270-277.

11. T.H. Cormen and D. Kotz, "Integrating Theory and Practice in Parallel File Systems," *Proc. 1993 Symp. Dartmouth Inst. for Advanced Graduate Studies and Parallel Computing*, Dartmouth College, Hanover, N.H., 1993, pp. 64-74.

12. *High-Performance Fortran Language Specification*, Version 0.3, CRPC Tech. Report, High-Performance Fortran Forum, Rice Univ., 1992, pp. 6-7.

Juan Miguel del Rosario is studying for his PhD at Syracuse University and is a research assistant at the university's Northeast Parallel Architectures Center. His primary areas of interest are parallel computer architectures and parallel I/O systems, parallel operating systems, languages, and theoretical aspects of parallel computation and systems.

Del Rosario earned a BS in mathematics, physics, and chemistry in 1988 and an MS degree in computer science in 1992, both from the University of San Francisco. He is an IEEE Computer Society and ACM student member.

Alok N. Choudhary joined the Department of Electrical and Computer Engineering at Syracuse University in 1989 and is now an associate professor. His main research interests are parallel and distributed processing; software development environments for parallel computers, including compilers and runtime support; parallel computer architectures; and parallel I/O systems.

Choudhary received a BE in electrical and electronics engineering with honors from Birla Institute of Technology and Science in Pilani, India, in 1982; an MS from the University of Massachusetts, Amherst, in 1986; and a PhD in electrical and computer engineering from the University of Illinois, Urbana-Champaign, in 1989. He received the National Science Foundation Young Investigator Award in 1993. He is an IEEE Computer Society and ACM member.

Readers can contact Del Rosario at the Northeast Parallel Architectures Center, Syracuse University, 111 College Place, RM 3-201, Syracuse, NY 13244-4100, e-mail mrosario@nova.cat.syr.edu; and Choudhary at the Dept. of Electrical and Computer Engineering, Syracuse University, 121 Link Hall, Syracuse, NY 13244-1240, e-mail choudhar@cat.syr.edu.