Encyclopedia of Computer Science and Technology, Supplement 25, 1999.
ISBN13: 9780824722937.

I/O in Parallel and Distributed Systems

David Kotz
Department of Computer Science
Dartmouth College
Hanover, NH 03755-3510
dfk@cs.dartmouth.edu

Ravi Jain
Bellcore
445 South St
Morristown, NJ 07960

March 1, 1998

Abstract

We sketch the reasons for the I/O bottleneck in parallel and distributed systems, pointing out that it can be viewed as a special case of a general bottleneck that arises at all levels of the memory hierarchy. We argue that because of its severity, the I/O bottleneck deserves systematic attention at all levels of system design. We then present a survey of the issues raised by the I/O bottleneck in six key areas of parallel and distributed systems: applications, algorithms, languages and compilers, run-time libraries, operating systems, and architecture.

1 Introduction

I/O for parallel and distributed computer systems has drawn increasing attention over the past decade as it has become apparent that I/O performance, rather than CPU performance, may be the key limiting factor in the performance of future systems. This has led to a growing and systematic study of the *I/O bottleneck* in parallel and distributed systems.

The I/O bottleneck arises for three main reasons. First, while the speeds of CPUs have been increasing dramatically in the past few decades, the speed of I/O devices, being limited by the speed of mechanical components like disks and arms, has been increasing at a much slower rate. For example, while CPU speeds have been increasing at 50-100% per year, magnetic disk access time has decreased by only about one third in ten years [PH90], and these trends are likely to remain qualitatively unchanged. Second, in parallel and distributed systems multiple CPUs are

An earlier version of this text appeared as Chapter 1 in *I/O in Parallel and Distributed Systems*, Kluwer Academic Publishers, 1996.

employed simultaneously, thus exacerbating this speed mismatch. Finally, new application domains, such as multimedia, visualization, and Grand Challenge problems, are creating ever-increasing I/O demands. Gibson [Gib92] provides a historical review of the I/O bottleneck, as well as a discussion of the underlying technology trends.

While the I/O bottleneck remains a central concern for specialized application domains characterized by highly I/O-intensive programs, since the late 1980s the concern has spread to general-purpose supercomputers as well as lower-end machines [SHH90, Gib92, MK91, Cat92]. The increasing concern is linked to the increasing importance of I/O to overall system performance, due to the technological trends we discuss above. Indeed, some have argued that the performance of a supercomputer system should be measured in terms of its data-transfer rates, both within the system and across a network, instead of the peak floating-point computation rate [SHH90, Jor92].

It is also important to note that, to varying degrees, the I/O bottleneck exists at multiple levels of the memory hierarchy. While most work on the impact of I/O on system performance has focused on the widening gap between CPU and disk speeds, recently there has been renewed interest in the speed increase mismatch at other levels of the memory hierarchy.

One such area of concern is at the cache-DRAM interface [WM95]. While cache memory speeds are increasing rapidly (particularly for on-chip caches), the speed of DRAM is increasing much less rapidly. Suppose that the cache access time equals one instruction cycle, that DRAM is currently four times slower than cache, that the only cache misses are the compulsory ones where the data being accessed has never been read before, and that the compulsory miss rate is 1%. Assuming that on-chip cache speed is increasing at the same rate as CPU speed, i.e., 80% per year [Bas91], while DRAM speed increases at 7% per year [PH90], a DRAM access will take 1.52 CPU cycles in the year 2000, 8.25 cycles in 2005 and 98.8 cycles in 2010 [WM95]. In this scenario, system performance is determined not by the CPU speed but by memory speed, i.e., it hits the "memory wall". Changing some of the assumptions of the scenario (e.g., the current DRAM/cache speed ratio or the cache miss ratio) does not affect the overall trend as long as there is a mismatch in the *rate* of growth of DRAM and cache speeds, and the miss ratio is non-zero.

We view the memory wall as another manifestation of the bottleneck created whenever, like I/O, the rate of speed increase between two interfaced technologies is significantly mismatched, and it is not feasible to use the faster technology alone. It is possible that fundamental technical changes will take place, so that either the rate of growth of the faster technology decreases (e.g., by encountering new constraints at the sub-atomic level), or that of the slower technology increases (e.g., a new

secondary storage technology that is as cheap, fast, robust, dense, efficient, and relatively low in heat production as magnetic disk but with much greater potential for performance growth). Until this occurs, however, we believe that the I/O bottleneck calls for an integrated response from system designers and architects. We believe that the likely growth and severity of the I/O bottleneck for parallel and distributed systems demands attention at all levels of the system design, including applications, algorithms, compilers, operating systems, and architecture. These solutions should be scalable, as the size of the systems and applications grow, and as technology changes. Furthermore, the best solutions will likely be found in a comprehensive, system-wide approach to the problem.

In the following section we briefly survey some of the issues raised and solutions developed in recent years at several levels of system design, namely applications, algorithms, compilers, operating systems and architecture. Finally, we summarize in section 3.

2 Survey of I/O issues

In the last few years, there has been a surge of interest in addressing the parallel I/O bottleneck as different communities have discovered its effects. In this section we briefly discuss some of the issues that have been raised and some of the solutions proposed.

2.1 Applications

There have been two major application domains where I/O in parallel computer systems has traditionally been found to be a bottleneck. One is scientific computing with massive datasets, such as those found in seismic processing, climate modeling, and so forth [dC94, Kot96a]. The second is databases [DG92, BDLJ85].

The I/O bottleneck continues to be a serious concern for scientific computing, particularly Grand Challenge problems, where it is now commonly recognized as an obstacle [Sha95]. Many scientific applications generate 1 GB of I/O per run [CHKM96, dC94, MK91], and applications performing an order of magnitude more are not uncommon. Recent work by Acharya et al [AUB+96] describes an earth-science application, called `pathfinder`, which moves 28 GB of data; applications in computational physics and fluid dynamics are projected to require I/O on the order of 1 TB [dC94]. It seems clear that these total I/O requirements will keep increasing as scientists continue to study phenomena at larger space and time scales, and at finer space and time resolutions. Since the response time that humans can tolerate for obtaining computational results— no matter how

comprehensive and detailed— is always bounded, the I/O rates required will continue to increase also. Thus while current applications require I/O rates of tens of MBps for secondary storage, in the near future they will require I/O rates in the region of 1 GBps for secondary storage [dC94].

A similar trend can be seen in the area of databases, particularly for applications such as data mining [DG92]. New applications, such as mapping the human genome, turn out to involve large-scale database searches on gigabytes of data, and eventually terabytes of data.

Meanwhile, new classes of applications that are rapidly becoming ubiquitous are image visualization [AL92, Spe94] and multimedia information processing [Spe91, Spe95]. It seems likely that multimedia information will be found in many, if not all, computing environments in the future [Spe93, Pas93]. Multimedia information systems not only impose much higher throughput demands than traditional computer applications (e.g., 81 MBps for HDTV, or 100 MBps for 200 concurrent 0.5 MBps MPEG video streams from a video server) but also introduce additional constraints, such as real-time and synchronized data transfers [GVK+95], not found in the traditional applications.

Studies of the I/O behavior of applications have shown that applications vary very widely in their I/O characteristics. Looking at scientific applications alone, the I/O volume per MFLOP of computation has been found to vary from zero bytes to 8 KB [CHKM96]. Detailed studies of three scalable, I/O-intensive scientific applications (electron scattering, terrain rendering, and quantum chemistry), show tremendous variations in I/O workload parameters such as I/O request sizes and the total I/O volume [CACR95, SACR96].

It is clear that more work remains to be done in understanding the I/O characteristics of applications. Some efforts, such as the CHARISMA project [KN95], studied production scientific-application workloads. So far, however, there has been relatively little attention paid to the detailed I/O characteristics of non-scientific applications (e.g., visualization and multimedia databases). In any case, the study of existing I/O-intensive programs is difficult: since current parallel and distributed computer systems have limited I/O bandwidth, programmers write programs to circumvent these limitations. For example, the REACT code for quantum chemical reaction dynamics simulation encounters I/O bandwidth limits, so users recalculate rather than store and retrieve data [CHKM96].

We believe that parallel and distributed applications should be designed to deal directly with the I/O bottleneck. Acharya et al [AUB+96] describe results from running four scientific applications on a 16-processor IBM SP-2 machine with up to six fast disks attached to every processor. Although the aggregate disk bandwidth was 768 MBps and the aggregate bus bandwidth was 480 MBps, it

was found that application-level I/O bandwidth achieved when reading and writing files was only about 15-24 MBps. After rewriting these applications to tune them to the system, application-level I/O rates of over 100 MBps were achieved. These results illustrate the benefits of designing applications for efficient I/O. Given the large variability observed in application I/O behavior, however, automated tools to help designers in their task are desirable.

2.2 Algorithms

An important consideration for any solution that addresses the I/O bottleneck is that the fundamental algorithms used by applications be asymptotically efficient in terms of the I/O activity they generate.

For sequential computers, the asymptotic efficiency of algorithms is considered in terms of the RAM model of computation. The success of the RAM model is based on its ability to realistically capture the fundamental characteristics of a wide range of sequential machines, while remaining sufficiently abstract to be tractable in the design and analysis of algorithms. The model assumes, however, that the data required by the computation is available in the main memory of the machine before the computation begins. This assumption is reasonable if the memory is large enough to hold the data, or the time required for I/O is small relative to the time required for computation. It has long been recognized that for many applications and system architectures, neither assumption holds, particularly as the size of data sets increases. Furthermore, the RAM model assumes that all memory locations are accessible at equal cost, which is increasingly not true in today's multi-level cache-memory hierarchies. Nonetheless, the RAM model remains a good first approximation for the analysis of many in-core algorithms on sequential machines.

When the application's data do not fit in main memory, however, the situation is quite different. The I/O time may dominate the performance of the application, so the RAM model is a relatively useless tool for analyzing performance. In the following we discuss some models of I/O complexity that have been developed; see Shriver and Nodine [SN96] for more details. We also discuss some of the issues these models raise.

In the *unrestricted parallel model* [AV88] the computer system is modeled as a single CPU connected to a main memory capable of holding M records, which in turn is connected to an external memory (disk) capable of storing at least the N records that are the data set of the algorithm. Any D blocks, each consisting of B contiguous records, can be transferred in a single I/O operation, and it is assumed that $1 \leq DB \leq M \leq N$. The unrestricted parallel model is a

two-level model of memory hierarchy, unlike the single level of memory in the RAM model, and captures two different forms of parallelism in the data transfer: block transfer, since a single I/O operation transfers B records simultaneously, and parallel disk transfer, since D blocks can be transferred simultaneously. Block transfers are quite important in practice, since disk seek times often dominate the total I/O time of a block. The model is termed unrestricted since *any* D blocks can be transferred simultaneously.

The *parallel disk model* [VS90, VS94a, VS94b] extends the unrestricted parallel model by requiring the D parallel block transfers to be from D separate disks, in which consecutive blocks are stored on consecutive disks, and an I/O operation can transfer at most one block per disk. The parallel disk model is clearly more realistic than the unrestricted parallel model, which allows the algorithm designer to ignore the critical issue of partitioning and allocating a large data set across multiple disk drives so as to balance disk loads. Numerous algorithms have been developed using the parallel disk model (PDM), from sorting and permutation primitives to computational geometry [SN96, AVV97].

Nonetheless, the parallel disk model still seems limited in that it does not model the use of caching. Several models have been developed for dealing with multi-level memory hierarchies. One is the *Uniform Memory Hierarchy* (UMH) model [VN93]. The model can be adjusted using two integer parameters, α and ρ ($\alpha, \rho \geq 2$). The memory consists of multiple levels, where the lth memory level consists of $\alpha\rho^l$ blocks, each of size ρ^l, and is connected via buses to levels $l+1$ and $l-1$, where the bandwidth of the bus between level l and $l+1$ is given by some function $b(l)$. Any block at any level l can be randomly accessed and transferred to or from level $l+1$ in time $\rho^l/b(l)$. The CPU resides at level 0. While the UMH model seems more realistic than the parallel disk model, it is harder to analyze algorithms in this framework.

An important factor not captured by I/O complexity models is the ability to overlap I/O with computation. Several techniques have been developed that attempt to obtain this overlap, such as the use of write-behind caching policies, log-structured file systems, user-level data prefetching hints [PGG+95], and language and compiler techniques [RBC91].

The development of these techniques, as well as the observation of the usefulness of such overlap in practice [WGWR93], underlines the importance of validating the theoretical models of I/O complexity against careful experiments. As noted by Shriver and Nodine [SN96], despite the numerous algorithms that have been developed for these models, researchers have only recently attempted experimental validation of the model or algorithms. Vengroff developed a programmer's library

based on the PDM, called TPIE, and implemented several algorithms [Ven96]. In Cormen's experiments involving permutations on out-of-core data [CH97], he found that many applications built using the parallel disk model (PDM) are not I/O bound and thus performance does not necessarily track the PDM parameters, particularly D and B. It was found to be particularly important to overlap I/O and computation.

Furthermore, the performance of applications running on parallel computers is often dependent on parameters ignored by the PDM, particularly inter-processor communication time. For example, Womble and others [WGWR93] found that the total time taken by their LU factorization algorithm on the nCUBE 2 did increase as the amount of main memory available decreased, as predicted by the parallel disk model. This increase was almost entirely due to the fact that the computation's grain size decreased, however, resulting in increased interprocessor communication costs, rather than an increase in I/O. More recently, Cormen and others [CWN97] implemented and measured several methods to perform out-of-core Fast-Fourier-Transform (FFT) computations. They developed new approaches to computing the FFT that are based on PDM algorithms for out-of-core permutations. In their experiments they found that inter-process communication was a significant component of overall performance, often the determining factor in comparisons between different algorithms.

These results indicate the importance of further experimental work to validate the underlying I/O models and discover the regimes where they are most useful in practice. Hopefully, this experimental work will help to define a new, tractable model that permits I/O, computation, and communication to be considered at the same time, and allow development of algorithms that balance the time taken by these three basic activities. Perhaps a model that allows representation of these different activities to different degrees of precision, to match the characteristics of the problem and the architecture under consideration, would overcome this difficulty.

2.3 Language and compiler support

In addition to developing specialized I/O-efficient algorithms for fundamental operations such as sorting, it is important to extend parallel compiler technology to automatically generate I/O-efficient code for important classes of applications.

Reddy et al have argued [RBC91] that without significant advances in I/O compilation techniques, the parallel I/O hardware offered by many commercial architectures may be of little use in reducing the total execution times of individual programs. Several system-wide techniques for improving I/O speeds have been designed, such as reducing I/O due to paging, and prefetching

pages and cache lines [MDK96]. Although these techniques are helpful, they cannot exploit the domain-specific characteristics of individual programs. Unless programmers are provided higher-level language constructs for specifying the I/O requirements and access patterns of their programs, however, it is difficult and tedious to manually optimize program I/O to take advantage of domain-specific information.

There have been several efforts at extending existing languages to provide constructs that allow a compiler to attempt to optimize the I/O of individual programs. Typically these languages, like HPF [Hig93], Fortran D [FHK+90] and ViC* [CC94] assume a data-parallel programming model, in which the same sequence of operations is to be applied to all the elements of a large data structure (e.g., an array or vector). The data-parallel programming model is well suited to many types of regular scientific computations.

In data-parallel languages like HPF, the user can issue compiler directives that specify how data arrays are to be partitioned for parallel access by multiple processors. The directives include constructs for specifying common data alignment and access methods (e.g., block, cyclic, block-cyclic). The compiler uses these directives to partition the computation and to generate the appropriate communication and synchronization code for permitting parallel data access. Bordawekar and Choudhary [BC96] provide further details on issues in compiling I/O-intensive problems expressed in data parallel languages.

While the data-parallel paradigm is undoubtedly important and widespread, many parallel applications do not have a regular structure. There has been a lot of activity in designing computational techniques for irregular parallel computations, particularly in scientific applications such as computational fluid dynamics. So far there has been little attention paid to developing compiler support for the I/O behavior of irregular parallel computations. Similarly, high-level parallel languages and programming environments often do not contain support for specifying the I/O activities of the program, although the recent incorporation of I/O constructs into the MPI message-passing standard is promising [MPI97].

2.4 Run-time libraries

Perhaps one of the most active areas of software development in parallel-I/O research has been in run-time libraries. Run-time libraries are the quickest way to provide I/O support to a wide range of applications on a wide range of platforms, avoiding the need to modify compilers or operating systems. Several libraries have been developed to support applications wishing to encode

algorithms from the parallel disk model, particularly TPIE [Ven96]. Some are designed to support a compiler, such as ViC* [CC94, CH97] or PASSION [TCB+96]. Others are oriented toward scientific applications in general [TG96, SW96]. Still others are designed for specific application domains, such as computational chemistry [NFK98]. Although few attempt to be standards, except MPI-2 [MPI97], these libraries allow the application programmer to take advantage of carefully tuned algorithms and proven techniques.

2.5 Operating systems

The operating system of a parallel or distributed computer has to strike a delicate balance when it comes to I/O. On one hand, it must provide the programmer facilities that ease the programming task and hide the details and complexities of coordinating and efficiently utilizing the underlying I/O hardware and devices; on the other hand, it must allow the programmer sufficient control to efficiently use the rich resources of the system.

Since I/O performance is an increasingly important component of overall application performance, and since a large number of I/O-optimal algorithms for fundamental operations have been developed, it is important that the operating system allow these algorithms to be utilized. Cormen and Kotz [CK93] argue that to allow the use of I/O-optimal algorithms, the parallel computer system must have the following capabilities: it must allow the algorithm to control the declustering of files, query about the system configuration, perform independent parallel disk I/O, turn off parity (for systems such as RAID [PGK88]), and turn off caching and prefetching. Until very recently, most current operating systems did not provide the programmer with these capabilities.[1]

There has been a recent surge of activity in parallel file systems [CF96, KS97, NK97, MK97, HER+95] and parallel I/O interfaces [MPI97, CPD+96] that address some of these requirements. One common feature of many parallel file interfaces is that the programmer to specify the access pattern for each file. Typically, this access pattern is specified with one or more parameters (sometimes called *modes*, *filetypes* [MPI97], or *templates* [PUSS97]). Then, when the usual file read and write operations are invoked by multiple processors, the semantics of the operation (and the actual set of file bytes to read and write) are determined by the declared file mode. The most common modes can be classified as [CFF+96]

[1] Note that it is possible to use RAIDs with I/O-optimal algorithms. In a situation where there are multiple RAIDs attached to the parallel computer, each RAID can be considered to be a single, high-performance disk from the point of view of the algorithm. In that case, each RAID performs parity and synchronized, fully-striped I/O internally, at a level not visible to the algorithm. While this technique works, it does not expose the parallelism of the system to the application, possibly limiting performance.

- *Broadcast-reduce,* where all processes collectively access the same data,
- *Scatter-gather,* where all processes collectively access a sequence of data blocks, in order,
- *Shared offset*, where all processes operate independently but share a common file pointer, and
- *Independent*, where the programmer is allowed complete freedom to specify access.

Other studies show that most common parallel file-access patterns can be captured in terms of simple file partitioning schemes and access modes [NK96, NKP+96]. Determining the right set of abstractions is still an issue, however. Some file systems like Vesta [CF96] and Panda [SW96] allow the programmer to choose parameters that determine the mapping of file data to disks, and file accesses to file data. HFS takes a different approach, allowing the programmer to construct a file abstraction from object-oriented components, such as a distribution component and a replication component [KS97]. The Galley file system [NK97] and the SIO interface [CPD+96] choose a low-level philosophy, in which the file system provides only a low-level interface with highly flexible primitives. Other interfaces and abstractions, such as MPI-2, Vesta, Panda, or HFS, can be built on top of these lower-level interfaces. Some believe that even more flexibility should provide still more control to the library programmer [KN96].

Underneath the interface, much of the research in parallel file systems involves techniques for high-performance implementations. Techniques like disk-directed I/O [Kot97], caching and prefetching [ACR95, KTP+96, KE93, MDK96], and access-pattern classification [MR97]. These techniques can often lead to orders of magnitude better performance.

Much of the work on parallel file systems has been oriented towards the traditional staple of parallel computing, namely scientific applications. On the other hand, some features of parallel computer operating systems may not be well-suited to support high data-rate applications such as multimedia information systems. One such feature is the large amount of data copying that takes place to perform data transfers [FP93, PAM94, Ste94, KSU94]. Thus it is not unusual for a single transfer from an I/O device to an application process to involve a copy from the device to the device I/O buffer, another copy from the device buffer to a kernel buffer, and a third copy to a user process buffer; the sequence of copies may be repeated in reverse for a process-to-device copy. While the nominal bandwidths for most workstation buses are 100 MBps or more, measured bandwidths for copying un-cached data are almost an order of magnitude less. Repeated data copying further reduces the effective bus throughputs and severely impacts the response times for

applications such as digital video and audio (DVA). Bypassing some of this copying can produce significant performance improvements.

Separating the data from the control information about the data (or "meta-data") can help bypass some copying and reduce CPU involvement by allowing some transfers to be done by DMA [Pas92, Pas93, CW93].

Separating data from control information is also highly desirable to perform effective data-transfer scheduling. Jain and others have developed centralized and distributed scheduling algorithms that can take advantage of such control information [JWBS92, Jai93, JSWB92, JSWB97, DJT96]. These algorithms are operating system-level, parallel-I/O scheduling algorithms, i.e., they are intended for use in an operating system that handles I/O requests for multiple applications, in systems where multiple I/O transfers can take place simultaneously. Thus they differ from traditional disk-scheduling algorithms (which schedule disk arm movements at the level of individual disks) and the application-specific I/O-scheduling algorithms (which schedule I/O operations of individual programs, e.g., for out-of-core sorting [VS90, VS94a, VS94b]). Parallel-I/O scheduling is required because even if each individual disk schedules I/Os to minimize arm movement, and each individual application issues a minimal number of I/O requests, the simultaneous I/O requests of multiple applications for data residing on multiple disks can result in conflicts that, unless properly resolved via scheduling, can result in long delays and inefficiencies. We expect that as parallelism in the I/O subsystem becomes more common, and multiple applications running on multiprocessors become the norm, parallel-I/O scheduling will become increasingly important.

2.6 Architecture

Possibly the area that has received the most attention in terms of the I/O bottleneck has been the disk subsystem architecture. The use of *low-level parallelism* in the service of I/O in schemes such as disk interleaving, striping, RAID, RADD, etc., is well known [Kim86, PGK88, Gib92, SS90] and is briefly reviewed by Kotz [Kot96b]. The gains provided by the low-level schemes can be overwhelmed, however, unless scalable algorithms, smart compilers and appropriate operating systems mechanisms are used to increase the I/O parallelism at higher levels of the system; these higher-level techniques in turn lead to additional requirements upon the architecture. For instance, the use of I/O-optimal algorithms for sorting and matrix multiplication imposes architectural requirements, such as the ability to perform independent parallel disk accesses and to turn off parity, that are typically not supported by architectures employing such low-level schemes [CK93]. In particular,

if disk striping is used, where the read/write heads of all the disk drives move synchronously, the I/O complexity of the optimal algorithms increases by more than a constant factor [VN93].

The continued bottleneck in I/O performance despite the use of these low-level schemes has led to proposals that closer attention be paid to the I/O interconnection architecture [GGD93, YM96, and references therein]. Several hypercube parallel computer system designs rely upon I/O nodes embedded at selected nodes of the hypercube. Ghosh and others [GGD93] note that the interprocessor communication links are used for both I/O and interprocessor traffic, and that overlapping the two traffic types can reduce performance due to congestion. They propose that a separate network be used for interconnecting the I/O nodes; simulations show that not only can response times and latencies be improved, but the performance can be made relatively insensitive to data locality.

The emergence of multimedia applications such as digital video and audio (DVA) also motivates further architectural requirements. Pasquale [Pas93] argues that the high data rates and timing constraints of DVA necessitate better control over low-level timing of I/O transfers. In particular, system I/O channels should be interruptable and it should be possible to schedule the data transfers across the channels. It should also be possible to perform DMA to and from all devices and memory, using all addresses (i.e., not just word-aligned or block-aligned addresses). Although some buses like the IBM Microchannel do have this capability, they are typically not used in this fashion. Device controllers should be capable of large-grained burst-mode transfers, and should have relatively large memory buffers to help smooth out the jitter between different media streams [Pas93, Ste90].

3 Summary

We sketch the reasons for the I/O bottleneck in parallel and distributed systems, pointing out that it can be viewed as a special case of a general bottleneck that arises at all levels of the memory hierarchy. We argue that because of its severity, the I/O bottleneck deserves systematic attention at all levels of system design. We then survey the issues raised by the I/O bottleneck in six key areas of parallel and distributed systems: applications, algorithms, compilers and languages, run-time libraries, operating systems, and architecture. We summarize these below.

We observe that the I/O bottleneck continues to be a serious concern for scientific computing and database applications, and that the emerging areas of multimedia and visualization bring new I/O challenges. The I/O behavior of all application types needs to be studied in more detail, and more attention paid to designing applications to directly deal with the I/O bottleneck.

In the area of algorithms, we summarize the models of I/O complexity being used to develop I/O-efficient algorithms for important functions, such as sorting and FFT. The I/O models currently being used suffer from the drawback that they may not be realistic or representative of the I/O behavior of important applications on real machines. More work needs to be done to validate the I/O complexity models to determine the machine classes or operational regimes where they are appropriate. In addition, it is desirable to have models that capture computation, communication, and I/O in an integrated fashion, yet remain tractable.

Several efforts extend existing languages to provide constructs that allow a compiler to optimize the I/O of individual programs. Typically these efforts have focused on languages that assume a data-parallel programming model, which has been particularly useful for scientific computations. It would be useful to extend language and compiler support for the I/O performed by non-regular computations, or non-scientific applications, and to provide this support in higher-level (e.g., graphical) programming languages and environments.

There has been a recent surge of activity in parallel file systems and parallel-I/O interfaces for operating systems. Once again, much of this work has focused on support for scientific applications. There has been a growing awareness of the operating-system overheads of moving data, especially in terms of repeated copying of data as it is moved among various system and user buffers. Another important issue is to separate data from the control information about the data ('metadata') to allow better control and coordination of I/O activities, e.g., via scheduling.

Architectural solutions to the I/O bottleneck have enjoyed tremendous acceptance, particularly in the use of RAID and parallelism within the I/O subsystem. Given the persistence of the I/O bottleneck despite these solutions, the scope of architecture investigations needs to be broadened. There has been some work on alternative I/O interconnection architectures as well as the possibilities offered by the growing capabilities of individual subsystem components like disk controllers.

There remains much exciting research in the field of I/O for parallel and distributed systems. A forward-looking survey of the problems and prospects for research in the area was completed in 1996 [GVW96], and may be a useful starting point for researchers entering the field.

In this short overview of I/O in parallel and distributed systems we have necessarily omitted mention of many interesting research projects and papers. For more information, including pointers to many research projects, software packages, events, and related papers from the scientific literature, visit the parallel-I/O web page[2] and attend the Workshop on I/O in Parallel and Distributed

[2] `http://www.cs.dartmouth.edu/pario/`.

Systems (IOPADS[3]), held every 18 months.

References

[ACR95] Meenakshi Arunachalam, Alok Choudhary, and Brad Rullman. A prefetching prototype for the parallel file system on the Paragon. In *Proceedings of the 1995 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, pages 321–323, May 1995. Extended Abstract.

[AL92] Matthew Arrott and Sara Latta. Perspectives on visualization. *IEEE Spectrum*, 29(9):61–65, September 1992.

[AUB+96] Anurag Acharya, Mustafa Uysal, Robert Bennett, Assaf Mendelson, Michael Beynon, Jeffrey K. Hollingsworth, Joel Saltz, and Alan Sussman. Tuning the performance of I/O intensive parallel applications. In *Proceedings of the Fourth Workshop on Input/Output in Parallel and Distributed Systems*, pages 15–27, Philadelphia, May 1996. ACM Press.

[AV88] Alok Aggarwal and Jeffrey Scott Vitter. The input/output complexity of sorting and related problems. *Communications of the ACM*, 31(9):1116–1127, September 1988.

[AVV97] Lars Arge, Darren Erik Vengroff, and Jeffrey Scott Vitter. External-memory algorithms for processing line segments in geographic information systems. *Algorithmica*, 1997. To appear.

[Bas91] Forrest Baskett. Keynote address. International Symposium on Shared Memory Processing, April 1991.

[BC96] Rajesh Bordawekar and Alok Choudhary. Issues in compiling I/O intensive problems. In Jain et al. [JWB96], chapter 3, pages 69–96.

[BDLJ85] J. C. Browne, A. G. Dale, C. Leung, and R. Jenevein. A parallel multi-stage I/O architecture with self-managing disk cache for database management applications. In *Proceedings of the Fourth International Workshop on Database Machines*. Springer-Verlag, March 1985.

[3] `http://www.cs.dartmouth.edu/iopads/`.

[CACR95] Phyllis E. Crandall, Ruth A. Aydt, Andrew A. Chien, and Daniel A. Reed. Input/output characteristics of scalable parallel applications. In *Proceedings of Supercomputing '95*, San Diego, CA, December 1995. IEEE Computer Society Press.

[Cat92] Charles E. Catlett. Balancing resources. *IEEE Spectrum*, 29(9):48–55, September 1992.

[CC94] Thomas H. Cormen and Alex Colvin. ViC*: A preprocessor for virtual-memory C*. Technical Report PCS-TR94-243, Dept. of Computer Science, Dartmouth College, November 1994.

[CF96] Peter F. Corbett and Dror G. Feitelson. The Vesta parallel file system. *ACM Transactions on Computer Systems*, 14(3):225–264, August 1996.

[CFF+96] Peter Corbett, Dror Feitelson, Sam Fineberg, Yarsun Hsu, Bill Nitzberg, Jean-Pierre Prost, Marc Snir, Bernard Traversat, and Parkson Wong. Overview of the MPI-IO parallel I/O interface. In Jain et al. [JWB96], chapter 5, pages 127–146.

[CH97] Thomas H. Cormen and Melissa Hirschl. Early experiences in evaluating the parallel disk model with the ViC* implementation. *Parallel Computing*, 23(4):571–600, June 1997.

[CHKM96] Robert Cypher, Alex Ho, Smaragda Konstantinidou, and Paul Messina. A quantitative study of parallel scientific applications with explicit communication. *Journal of Supercomputing*, 10(1):5–24, March 1996.

[CK93] Thomas H. Cormen and David Kotz. Integrating theory and practice in parallel file systems. Technical Report PCS-TR93-188, Dept. of Math and Computer Science, Dartmouth College, March 1993. Revised 9/20/94.

[CPD+96] Peter F. Corbett, Jean-Pierre Prost, Chris Demetriou, Garth Gibson, Erik Reidel, Jim Zelenka, Yuqun Chen, Ed Felten, Kai Li, John Hartman, Larry Peterson, Brian Bershad, Alec Wolman, and Ruth Aydt. Proposal for a common parallel file system programming interface. WWW http://www.cs.arizona.edu/sio/api1.0.ps, September 1996. Version 1.0.

[CW93] S. A. Coleman and R. W. Watson. New architectures to reduce I/O bottlenecks in high-performance systems. In *Proceedings of the Twenty-Sixth Annual Hawaii International Conference on System Sciences*, page 5, January 1993.

[CWN97] Thomas H. Cormen, Jake Wegmann, and David M. Nicol. Multiprocessor out-of-core FFTs with distributed memory and parallel disks. In *Proceedings of the Fifth Workshop on Input/Output in Parallel and Distributed Systems*, pages 68–78, San Jose, CA, November 1997. ACM Press.

[dC94] Juan Miguel del Rosario and Alok Choudhary. High performance I/O for parallel computers: Problems and prospects. *IEEE Computer*, 27(3):59–68, March 1994.

[DG92] David DeWitt and Jim Gray. Parallel database systems: The future of high-performance database systems. *Communications of the ACM*, 35(6):85–98, June 1992.

[DJT96] Dannie Durand, Ravi Jain, and David Tseytlin. Improving the performance of parallel I/O using distributed scheduling algorithms. In Jain et al. [JWB96], chapter 11, pages 245–269.

[FHK+90] Geoffrey Fox, Seema Hiranadani, Ken Kennedy, Charles Koelbel, Uli Kremer, and Chau-Wen Tseng. Fortran D language specifications. Technical Report COMP TR90-141, Rice University, 1990.

[FP93] Kevin Fall and Joseph Pasquale. Exploiting in-kernel data paths to improve I/O throughput and cpu availability. In *Proceedings of the 1993 Winter USENIX Technical Conference*, pages 327–334, January 1993.

[GGD93] Joydeep Ghosh, Kelvin D. Goveas, and Jeffrey T. Draper. Performance evaluation of a parallel I/O subsystem for hypercube multiprocessors. *Journal of Parallel and Distributed Computing*, 17(1–2):90–106, January and February 1993.

[Gib92] Garth A. Gibson. *Redundant Disk Arrays: Reliable, Parallel Secondary Storage*. An ACM Distinguished Dissertation 1991. MIT Press, 1992.

[GVK+95] D. James Gemmell, Harrick M. Vin, Dilip D. Kandlur, P. Venkat Rangan, and Lawrence A. Rowe. Multimedia storage servers: A tutorial. *IEEE Computer*, 28(5):40–49, May 1995.

[GVW96] Garth A. Gibson, Jeffrey Scott Vitter, and John Wilkes. Strategic directions in storage I/O issues in large-scale computing. *ACM Computing Surveys*, 28(4):779–793, December 1996.

[HER+95] Jay Huber, Christopher L. Elford, Daniel A. Reed, Andrew A. Chien, and David S. Blumenthal. PPFS: A high performance portable parallel file system. In *Proceedings of the 9th ACM International Conference on Supercomputing*, pages 385–394, Barcelona, July 1995. ACM Press.

[Hig93] High Performance Fortran Forum. High Performance Fortran. *Scientific Programming*, 2(1–2):1–170, Spring-Summer 1993.

[Jai93] Ravi Jain. Scheduling data transfers in parallel computers and communications systems. Technical Report TR-93-03, Univ. Texas at Austin, Dept. of Comp. Sci., February 1993.

[Jor92] Harry F. Jordan. Scalability of data transport. In *Proceedings of the IEEE Scalable High Performnce Computing Conference*, pages 1–8, April 1992.

[JSWB92] Ravid Jain, Kiran Somalwar, John Werth, and J. C. Browne. Scheduling parallel I/O operations in multiple bus systems. *Journal of Parallel and Distributed Computing*, 16(4):353–362, December 1992.

[JSWB97] Ravi Jain, Kiran Somalwar, John Werth, and J. C. Browne. Heuristics for scheduling I/O operations. *IEEE Transactions on Parallel and Distributed Systems*, 8(3):310–320, March 1997.

[JWB96] Ravi Jain, John Werth, and James C. Browne, editors. *Input/Output in Parallel and Distributed Computer Systems*, volume 362 of *The Kluwer International Series in Engineering and Computer Science*. Kluwer Academic Publishers, 1996.

[JWBS92] Ravi Jain, John Werth, J. C. Browne, and Galen Sasaki. A graph-theoretic model for the scheduling problem and its application to simultaneous resource scheduling. In *ORSA Conf. on Computer Science and Operations Research: New Developments in their Interfaces*, January 1992. Available from Pergamon Press.

[KE93] David Kotz and Carla Schlatter Ellis. Practical prefetching techniques for multiprocessor file systems. *Journal of Distributed and Parallel Databases*, 1(1):33–51, January 1993.

[Kim86] Michelle Y. Kim. Synchronized disk interleaving. *IEEE Transactions on Computers*, C-35(11):978–988, November 1986.

[KN95] David Kotz and Nils Nieuwejaar. File-system workload on a scientific multiprocessor. *IEEE Parallel and Distributed Technology*, pages 51–60, Spring 1995.

[KN96] David Kotz and Nils Nieuwejaar. Flexibility and performance of parallel file systems. *ACM Operating Systems Review*, 30(2):63–73, April 1996.

[Kot96a] David Kotz. Applications of parallel I/O. Technical Report PCS-TR96-297, Dept. of Computer Science, Dartmouth College, October 1996. Release 1.

[Kot96b] David Kotz. Introduction to multiprocessor I/O architecture. In Jain et al. [JWB96], chapter 4, pages 97–123.

[Kot97] David Kotz. Disk-directed I/O for MIMD multiprocessors. *ACM Transactions on Computer Systems*, 15(1):41–74, February 1997.

[KS97] Orran Krieger and Michael Stumm. HFS: A performance-oriented flexible file system based on building-block compositions. *ACM Transactions on Computer Systems*, 15(3):286–321, August 1997.

[KSU94] Orran Krieger, Michael Stumm, and Ronald Unrau. The Alloc Stream Facility: A redesign of application-level stream I/O. *IEEE Computer*, 27(3):75–82, March 1994.

[KTP+96] Tracy Kimbrel, Andrew Tomkins, R. Hugo Patterson, Brian Bershad, Pei Cao, Edward Felten, Garth Gibson, Anna R. Karlin, and Kai Li. A trace-driven comparison of algorithms for parallel prefetching and caching. In *Proceedings of the 1996 Symposium on Operating Systems Design and Implementation*, pages 19–34. USENIX Association, October 1996.

[MDK96] Todd C. Mowry, Angela K. Demke, and Orran Krieger. Automatic compiler-inserted I/O prefetching for out-of-core applications. In *Proceedings of the 1996 Symposium*

on Operating Systems Design and Implementation, pages 3–17. USENIX Association, October 1996.

[MK91] Ethan L. Miller and Randy H. Katz. Input/output behavior of supercomputer applications. In *Proceedings of Supercomputing '91*, pages 567–576, Albuquerque, NM, November 1991. IEEE Computer Society Press.

[MK97] Ethan L. Miller and Randy H. Katz. RAMA: An easy-to-use, high-performance parallel file system. *Parallel Computing*, 23(4):419–446, June 1997.

[MPI97] MPI-2: Extensions to the message-passing interface. The MPI Forum, July 1997.

[MR97] Tara M. Madhyastha and Daniel A. Reed. Input/output access pattern classification using hidden Markov models. In *Proceedings of the Fifth Workshop on Input/Output in Parallel and Distributed Systems*, pages 57–67, San Jose, CA, November 1997. ACM Press.

[NFK98] Jarek Nieplocha, Ian Foster, and Rick Kendall. ChemIO: High-performance parallel I/O for computational chemistry applications. *International Journal of Supercomputer Applications and High Performance Computing*, 1998. To appear in a Special Issue on I/O in Parallel Applications.

[NK96] Nils Nieuwejaar and David Kotz. Low-level interfaces for high-level parallel I/O. In Jain et al. [JWB96], chapter 9, pages 205–223.

[NK97] Nils Nieuwejaar and David Kotz. The Galley parallel file system. *Parallel Computing*, 23(4):447–476, June 1997.

[NKP+96] Nils Nieuwejaar, David Kotz, Apratim Purakayastha, Carla Schlatter Ellis, and Michael Best. File-access characteristics of parallel scientific workloads. *IEEE Transactions on Parallel and Distributed Systems*, 7(10):1075–1089, October 1996.

[PAM94] Joseph Pasquale, Eric Anderson, and P. Keith Muller. Container shipping: Operating system support for I/O-intensive applications. *IEEE Computer*, 27(3):84–93, March 1994.

[Pas92] Joseph Pasquale. I/O system design for intensive multimedia I/O. In *Proc. IEEE Workshop on Workstation Operating Systems*, April 1992.

[Pas93] Joseph Pasquale. Systems software and hardware support considerations for digital video and audio computing. In *Proceedings of the 26th Hawaii International Conference on Systems Sciences*, page 15, January 1993.

[PGG+95] R. Hugo Patterson, Garth A. Gibson, Eka Ginting, Daniel Stodolsky, and Jim Zelenka. Informed prefetching and caching. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, pages 79–95, Copper Mountain, CO, December 1995. ACM Press.

[PGK88] David Patterson, Garth Gibson, and Randy Katz. A case for redundant arrays of inexpensive disks (RAID). In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 109–116, Chicago, IL, June 1988. ACM Press.

[PH90] David A. Patterson and John L. Hennessy. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 1990.

[PUSS97] Ian Parsons, Ron Unrau, Jonathan Schaeffer, and Duane Szafron. PI/OT: Parallel I/O templates. *Parallel Computing*, 23(4):543–570, June 1997.

[RBC91] A. L. Narasimha Reddy, P. Banerjee, and D. K. Chen. Compiler support for parallel I/O operations. In *Proceedings of the 1991 International Conference on Parallel Processing*, pages II:290–II:291, St. Charles, IL, 1991. CRC Press.

[SACR96] Evgenia Smirni, Ruth A. Aydt, Andrew A. Chien, and Daniel A. Reed. I/O requirements of scientific applications: An evolutionary view. In *Proceedings of the Fifth IEEE International Symposium on High Performance Distributed Computing*, pages 49–59, Syracuse, NY, 1996. IEEE Computer Society Press.

[Sha95] O. Sharp. The grand challenges. *Byte*, pages 65–72, February 1995.

[SHH90] J. E. Smith, W.-C. Hsu, and C. Hsuing. Future general purpose supercomputer architectures. In *Proceedings of Supercomputing '90*, pages 796–804, 1990.

[SN96] Elizabeth Shriver and Mark Nodine. An introduction to parallel I/O models and algorithms. In Jain et al. [JWB96], chapter 2, pages 31–68.

[Spe91] Special issue on digital multimedia systems. *Communications of the ACM*, 34(10), April 1991.

[Spe93] Special issue on interactive multimedia. *IEEE Spectrum*, 30(3), March 1993.

[Spe94] Special issue on visualization. *IEEE Computer*, 27(7), July 1994.

[Spe95] Special issue on multimedia systems and applications. *IEEE Computer*, 28(3), March 1995.

[SS90] Michael Stonebraker and Gerhard A. Schloss. Distributed RAID — A new multiple copy algorithm. In *Proceedings of 6th International Data Engineering Conference*, pages 430–437, 1990.

[Ste90] R. Steinmetz. Synchronization properties in multimedia systems. *IEEE J. Sel. Areas Comm.*, page 401, April 1990.

[Ste94] Peter A. Steenkiste. A systematic approach to host interface design for high-speed networks. *IEEE Computer*, 27(3):47–57, March 1994.

[SW96] Kent E. Seamons and Marianne Winslett. Multidimensional array I/O in Panda 1.0. *Journal of Supercomputing*, 10(2):191–211, 1996.

[TCB+96] Rajeev Thakur, Alok Choudhary, Rajesh Bordawekar, Sachin More, and Sivaramakrishna Kuditipudi. Passion: Optimized I/O for parallel applications. *IEEE Computer*, 29(6):70–78, June 1996.

[TG96] Sivan Toledo and Fred G. Gustavson. The design and implementation of SOLAR, a portable library for scalable out-of-core linear algebra computations. In *Proceedings of the Fourth Workshop on Input/Output in Parallel and Distributed Systems*, pages 28–40, Philadelphia, May 1996. ACM Press.

[Ven96] Darren Erik Vengroff. *The Theory and Practice of I/O-Efficient Computation.* PhD thesis, Department of Computer Science, Brown University, Providence, RI, April 1996.

[VN93] Jeffrey Scott Vitter and Mark H. Nodine. Large-scale sorting in uniform memory hierarchies. *Journal of Parallel and Distributed Computing*, 17(1–2):107–114, January and February 1993.

[VS90] Jeffrey Scott Vitter and Elizabeth A. M. Shriver. Optimal disk I/O with parallel block transfer. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing (STOC '90)*, pages 159–169, May 1990.

[VS94a] Jeffrey Scott Vitter and Elizabeth A. M. Shriver. Algorithms for parallel memory I: Two-level memories. *Algorithmica*, 12(2/3):110–147, August and September 1994.

[VS94b] Jeffrey Scott Vitter and Elizabeth A. M. Shriver. Algorithms for parallel memory II: Hierarchical multilevel memories. *Algorithmica*, 12(2/3):148–169, August and September 1994.

[WGWR93] David Womble, David Greenberg, Stephen Wheat, and Rolf Riesen. Beyond core: Making parallel computer I/O practical. In *Proceedings of the 1993 DAGS/PC Symposium*, pages 56–63, Hanover, NH, June 1993. Dartmouth Institute for Advanced Graduate Studies.

[WM95] Wm. A. Wulf and Sally A. McKee. Hitting the memory wall: Implications of the obvious. *Computer Architecture News*, 23(1):20–24, March 1995.

[YM96] Haruo Yokota and Yasuyuki Mimatsu. A scalable disk system with data reconstruction functions. In Jain et al. [JWB96], chapter 16, pages 353–372.