1

INTRODUCTION TO MULTIPROCESSOR I/O ARCHITECTURE

David Kotz

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

ABSTRACT

The computational performance of multiprocessors continues to improve by leaps and bounds, fueled in part by rapid improvements in processor and interconnection technology. I/O performance thus becomes ever more critical, to avoid becoming the bottleneck of system performance. In this paper we provide an introduction to I/O architectural issues in multiprocessors, with a focus on disk subsystems. While we discuss examples from actual architectures and provide pointers to interesting research in the literature, we do not attempt to provide a comprehensive survey. We concentrate on a study of the architectural design issues, and the effects of different design alternatives.

1 INTRODUCTION

As high-performance computers continue their stunning increases in computational performance, fueled in part by rapid improvements in processor and interconnection technology, I/O becomes an increasingly important component of overall system performance. This fact is especially true for parallel computers, where the combination of numerous processors boosts computational performance, leaving I/O as the serial bottleneck that limits scalability [2]. Indeed, many scientific and commercial applications have tremendous I/O requirements [20], both for moving data in and out of the parallel computer, as

The author is funded by NSF under grant number CCR-9404919, and by NASA Ames under agreement number NCC 2-849. The author speaks for himself and not for NSF or NASA.

well as for manipulating datasets too large to fit in primary memory. Thus, it is imperative that a parallel *I/O* architecture is provided to support the parallel *computational* architecture.

In this paper we survey some of the fundamental issues in parallel-I/O architectural design, using several architectures from the past and present as examples. We consider I/O to disks, tapes, external networks, and graphics, with an emphasis on disks. In general, our focus is on input to and output from the multiprocessor itself. Thus, we focus on internal disk subsystems, rather than on network-attached file servers. Most modern multiprocessors have internal disk systems, because they provide more effective performance (especially for small requests), are scalable, and are particularly useful to support "out-of-core" applications [26]. Most multiprocessors are also connected to an external mass-storage system, for long-term, high-capacity storage, which is one reason to be interested in a fast, parallel network connection.

2 REVIEW AND TERMINOLOGY

We assume that the reader is familiar with the fundamentals of I/O architecture, but we provide a quick review here (for a good introduction, see [55], chapter 9). Figure 1 shows a typical uniprocessor architecture. The CPU-memory bus tends to be of proprietary design, tuned for the particular CPU or memory. A *bus adapter* bridges between the proprietary CPU-memory bus and an I/O bus, typically based on a standard such as SCSI or PCI. *Controllers* connect the standard bus to specific I/O *devices* (disk, network, or graphics). The controllers are responsible for the low-level management of the device, interpreting standard I/O commands from the bus. In this way, the CPU vendor need only provide an adapter to a standard I/O bus, and the device vendor need only provide a controller to connect to a standard I/O bus. In some buses, such as SCSI, the controller is typically packaged with the device.

Note that peak I/O bandwidth, in any architecture, is limited by the slowest component [27]. Data from the disk(s) must flow through the I/O bus, the bus adapter, the memory bus, and into the memory. If the data is then sent to another processor across the network, the data must flow back out of the memory, across the memory bus, through the bus adapter, across the I/O bus, through the network interface, and across the network. Furthermore, an in-memory copy may be necessary to repackage the data. Thus, the data may flow through the CPU and its cache. Any of these components may be a

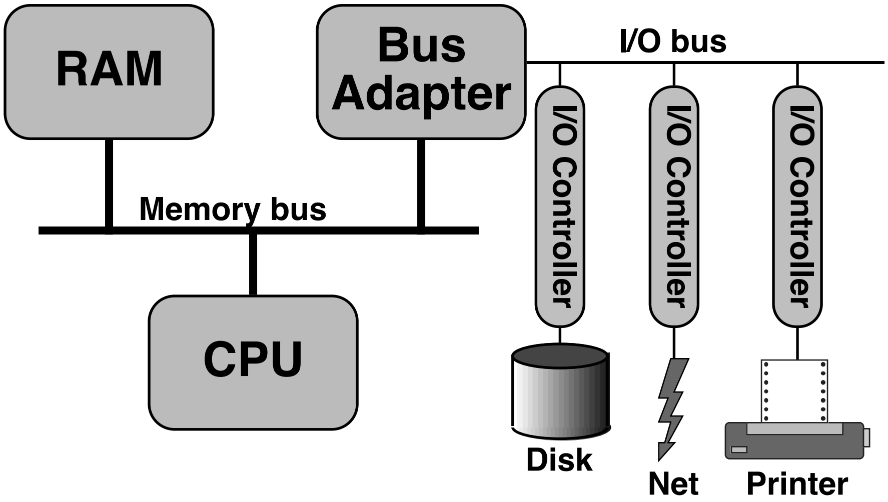


Figure 1 A typical uniprocessor architecture, showing the interconnection between processors and I/O devices, via the processor-memory bus, an adapter, an I/O bus, and a controller for each device.

bottleneck. Note also that the memory and memory-bus bandwidth needs to be 2–4 times that of the total disk or network bandwidth, because they are used more than once.

We also assume that the reader is familiar with the fundamentals of parallel-computer architecture (for an introduction see [1] or [55] chapter 10). In this paper we use Flynn's taxonomy [29] to distinguish between SIMD (single instruction stream, multiple data stream) and MIMD (multiple instruction stream, multiple data stream) architectures.

Among MIMD machines, we distinguish between multiple-address-space systems and shared-address-space systems (sometimes called shared-memory systems). In a multiple-address-space system, each processor has its own private physical address space, and the memory is physically distributed. Processors communicate explicitly by passing messages over an interconnection network. In a shared-address-space system, the hardware provides a shared physical address space. If the shared memory is physically centralized, we call it a Uniform Memory Access (UMA) architecture. If the shared memory is physically distributed, we call it a Non-Uniform Memory Access (NUMA) architecture. In either case, communication is implicit, with hardware translating accesses to remote addresses into messages on the interconnection network. Note that both architectures can support many different programming paradigms, including shared-memory and message-passing.

We often refer to processors, or processor-memory units, as “nodes,” a name that comes from a vision of processors as nodes in the graph of an interconnection network.

3 EXAMPLE ARCHITECTURES

We use the following machines as examples during our discussion of several issues in the design of parallel I/O architecture. Although there are many interesting parallel machines, we chose each of these as an interesting representative of an architectural category. We introduce each briefly below, and cover more details in later sections.

Shared-address-space UMA: DEC AlphaServer 2100

UMA (shared-memory) multiprocessors usually connect several CPUs to a single memory with a single bus. Today, small shared-memory multiprocessors are common, sold by nearly every Unix workstation vendor (they are sometimes called SMPs, for Symmetric MultiProcessors). In the simplest case, an UMA multiprocessor looks like the uniprocessor in Figure 1, but with multiple CPUs attached to the CPU-memory bus.

The DEC AlphaServer 2100 [59], sketched in Figure 2, includes at least three buses in a hierarchy. This structure allows connection of I/O devices designed either for the fast, new standard PCI bus or the slower, old standard EISA and SCSI buses. Since their PCI bus can sustain 132 MB/s, and one SCSI bus can handle 10-20 MB/s, it is possible to connect several SCSI buses to the PCI bus.

Shared-address-space NUMA: KSR 2

There are many different varieties of NUMA architecture, but perhaps the most recent common system is the KSR-2 [45]. Custom KSR microprocessors are interconnected by a hierarchy of rings, and specialized hardware manages nearly all of the memory in the machine as a shared cache, migrating sub-pages (cache lines) from processor to processor. A SCSI-bus adapter may be connected to any processor node.

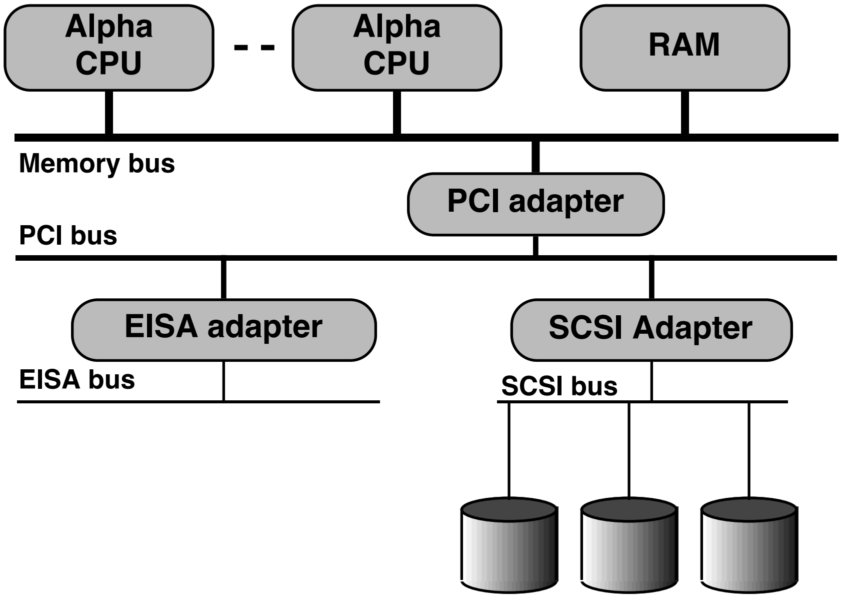


Figure 2 Architecture of the DEC 2100 AlphaServer, an UMA multiprocessor with substantial I/O capabilities. The DEC 2100 may be configured with up to 4 CPUs. (Adapted from [59] Figure 1.)

Other NUMA systems with interesting I/O architectures include the BBN Butterfly Plus [5], which had VME-bus adapters connected directly to the multistage omega interconnect, the NCR 3600 [50], with a tree interconnect and specialized I/O nodes at the leaves, and the Convex Exemplar [12], with a dedicated I/O processor for each cluster of computational processors.

Multiple-address-space, hypercube interconnect: nCUBE/ten

Some of the earliest large multiprocessors were based on a hypercube interconnect, and there have been many I/O studies specifically aimed at hypercube-interconnected multiprocessors [30, 32, 35, 58, 70]. Thus, we consider this class of machines separately from other multiple-address space machines.

We sketch the I/O architecture of the nCUBE/ten and nCUBE/2 in Figure 3 [19, 38, 57]. The nCUBE multiprocessor uses a hypercube topology

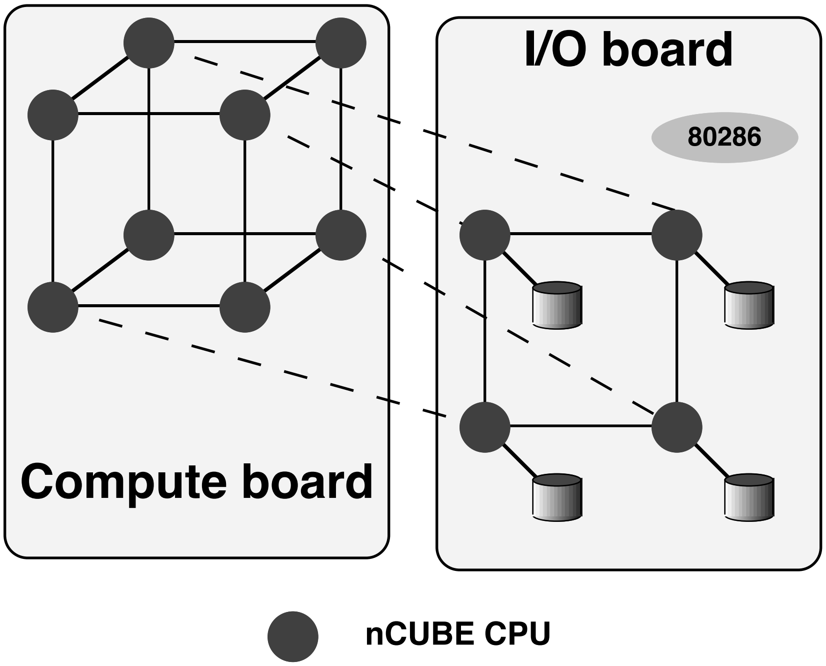


Figure 3 Simplified I/O architecture of the nCUBE/ten. The actual I/O board has 16 nCUBE CPUs, and the actual compute board has 64 nCUBE CPUs. Multiple boards are used to build larger systems. Memory is not shown; on the I/O board the nCUBE CPUs each share a region of memory with the 80286, which runs Unix and acts as a "host." The nCUBE/2 and nCUBE/3 are similar.

to interconnect custom microprocessors, each with several on-chip DMA ports used for the connection to neighboring processors. One large hypercube of processors is used for computation; a separate, smaller hypercube of processors is dedicated to I/O. These I/O processors are grouped onto boards of 16, along with an Intel 80286 CPU used as a "host" processor for interacting with users. An I/O controller (or SCSI adapter) may be connected to each of the I/O processors. Most of the on-chip ports on the I/O processors are used to connect to computational processors in the main hypercube.

The I/O architectures of the newer nCUBE/2 and nCUBE/3 [25] are similar to that of the nCUBE/ten, though of course they are larger and faster. The Intel iPSC machines are also similar, though without the smaller hypercube interconnecting I/O nodes, and without the 80286 host on the I/O boards [31, 56, 57].

Multiple-address-space, other interconnect: CM-5

Most recent multiple-address-space multiprocessors (including the IBM SP-2 and the Intel Paragon) Dedicate a subset of the nodes to I/O. These "I/O nodes" are the same type as the "compute nodes," with the addition of I/O buses and devices. The CM-5 [39, 65] is more interesting. The CM-5 is a collection of SPARC-based processor nodes connected by a fat-tree interconnect [47]. Some of the nodes are compute nodes, and some are dedicated I/O nodes, as shown in Figure 4 [46, 66]. Compute nodes are grouped into *partitions,* and each partition is assigned a special processor node as a *partition manager.* The I/O nodes are different from the compute nodes, and are specialized for different kinds of I/O devices: there are "disk" nodes, "HIPPI-network" nodes, and "tape" nodes. Each disk storage node (Figure 5) has a SPARC CPU as controller, a CM-5 network interface, an 8 MB buffer RAM, and four SCSI-2 bus adapters, typically with two disks each.

SIMD: Maspar MP-2

In the Maspar MP-2 [49], each processing element (PE) in the array is a simple 32-bit microprocessor with a small amount of memory. Figure 6 shows a sketch of the MP-2. All PEs execute instructions broadcast by the Array Control Unit (ACU), except for those PEs that may be temporarily *inactive* as a result of a conditional operation. The PEs are connected by three networks: a broadcast network for instructions from the ACU, a torus for nearest-neighbor communication, and a general "global router" for arbitrary inter-PE communication. The MP-2 adds I/O to the processor array by extending its global-router network to a separate I/O controller [51, 52]. Thus, a file-write operation becomes a global communication operation: all active PEs send data through the global router to the I/O RAM, which rearranges the data as necessary. The I/O controller then arranges disk access.

4 DISK I/O

In this section we discuss some of the architectural issues in parallel disk subsystems, and specific ways in which our example architectures deal with those issues. After a review of disk arrays, we focus on five fundamental issues in

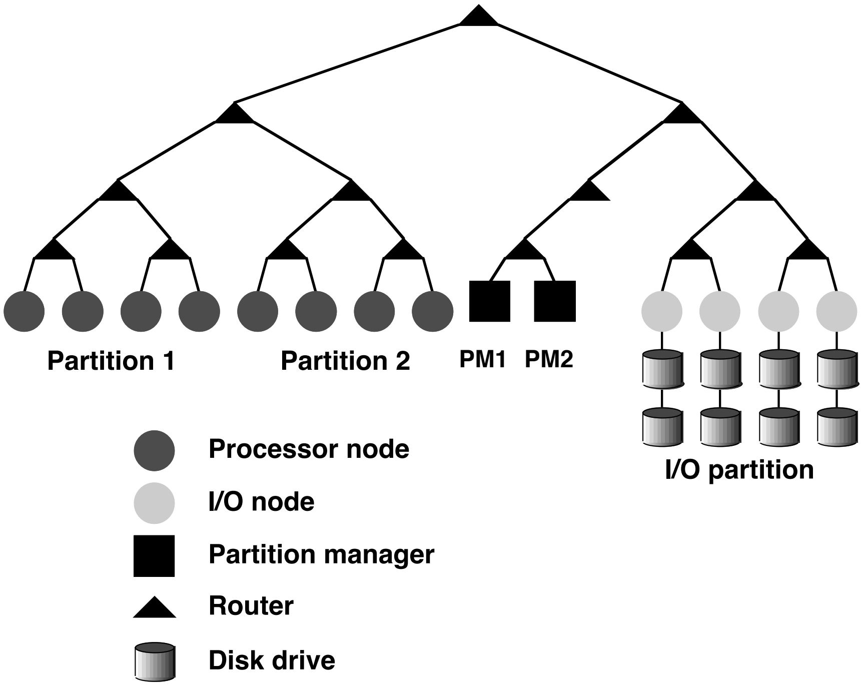


Figure 4 Typical attachment of I/O nodes in a CM-5 prevents I/O traffic from interfering with uninvolved partitions. In particular, inter-I/O-node transfers remain entirely within the I/O partition. All I/O traffic is managed by a partition manager (PM). (Adapted from [66].)

parallel-I/O architecture design: connection, management, placement, buffering, and availability.

4.1 Disk arrays and RAID

Although disk arrays are not the focus of this paper, they represent a fundamental form of parallel I/O. We thus review the topic of disk arrays and redundant disk arrays (RAID) for readers who may not be familiar with the topic. Chen et al. [13] and Gibson [33] provide more detailed surveys.

To improve the capacity and bandwidth of the disk subsystem, we may group several disks into a *disk array,* and distribute a file's data across all the disks in

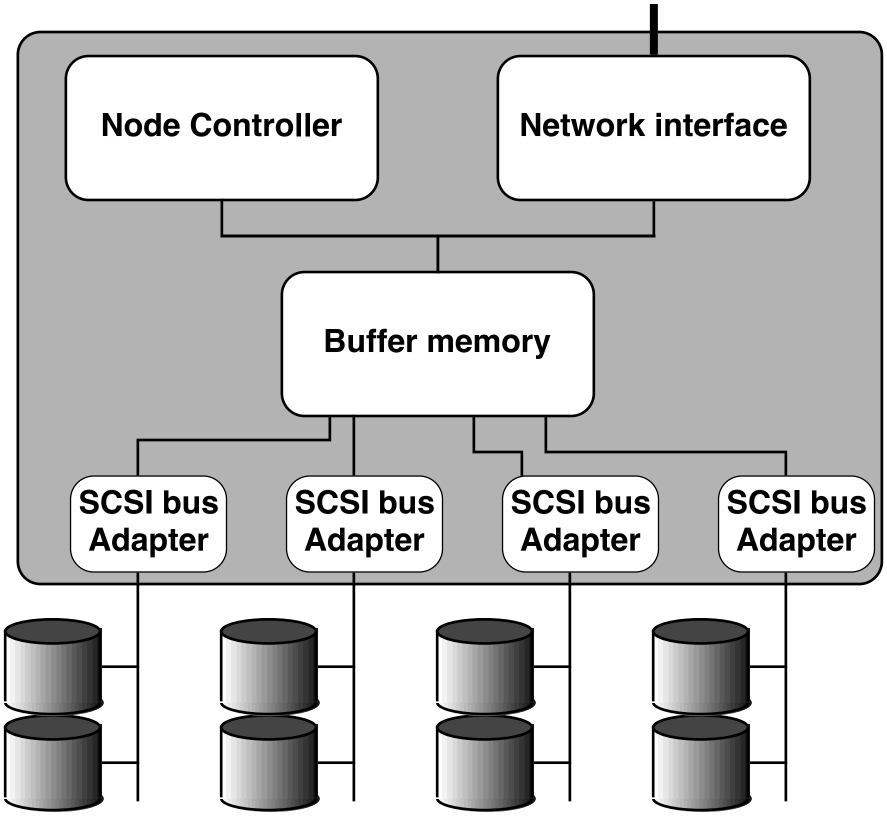


Figure 5 Architecture of a CM-5 disk storage node. (Adapted from [66].)

the group. This practice is typically called *striping*, *declustering,* or *interleaving*. There is no universal agreement on the definition of these terms, but common usage seems to indicate that *declustering* means any distribution of a file's data across multiple disks, whereas *striping* is a declustering based on a round-robin assignment of data units to disks. *Interleaving* is less commonly used now, but some have used it to mean striping when the disks are rotationally synchronized.

Early work by Kim [42] and Salem [60] demonstrated the usefulness of disk arrays, but one of the significant drawbacks was reduced reliability. Disk reliability is usually expressed in terms of the Mean Time To Failure (MTTF), with

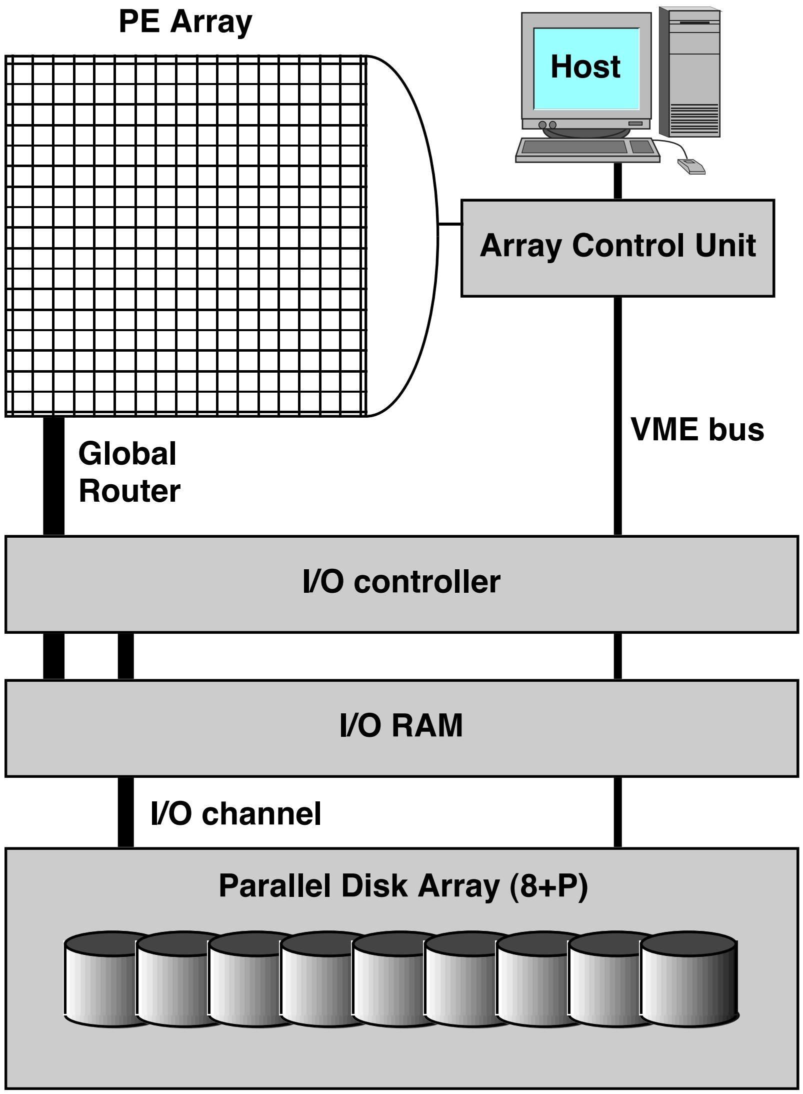


Figure 6 The Maspar MP-2 I/O architecture. The torus network is not shown. The global router allows general PE-to-PE communication, as well as communication between PEs and the *I/O RAM,* a large buffer memory. Individual global-router connections to each PE are not shown.

typical values in the hundreds of thousands of hours. If file data are striped across N disks, then the failure of any one disk essentially causes the loss of the file. If the disks are assumed to fail independently and with an exponential failure rate, then an N-disk array will fail (lose data) N times as often as a single disk, i.e., $\mathrm{MTTF}_N = \mathrm{MTTF}_1/N$. Some form of fault-tolerance is necessary to protect data against disk failure.

In 1988 Patterson, Gibson, and Katz presented "a case for redundant arrays of inexpensive disks (RAID)" [54], in which they argued that disk arrays could be faster, cheaper, smaller, and more reliable than traditional large disks, and categorized several techniques for using redundancy to boost the availability of disk arrays. We summarize the work here. Their RAID "levels" are (Figure 7):

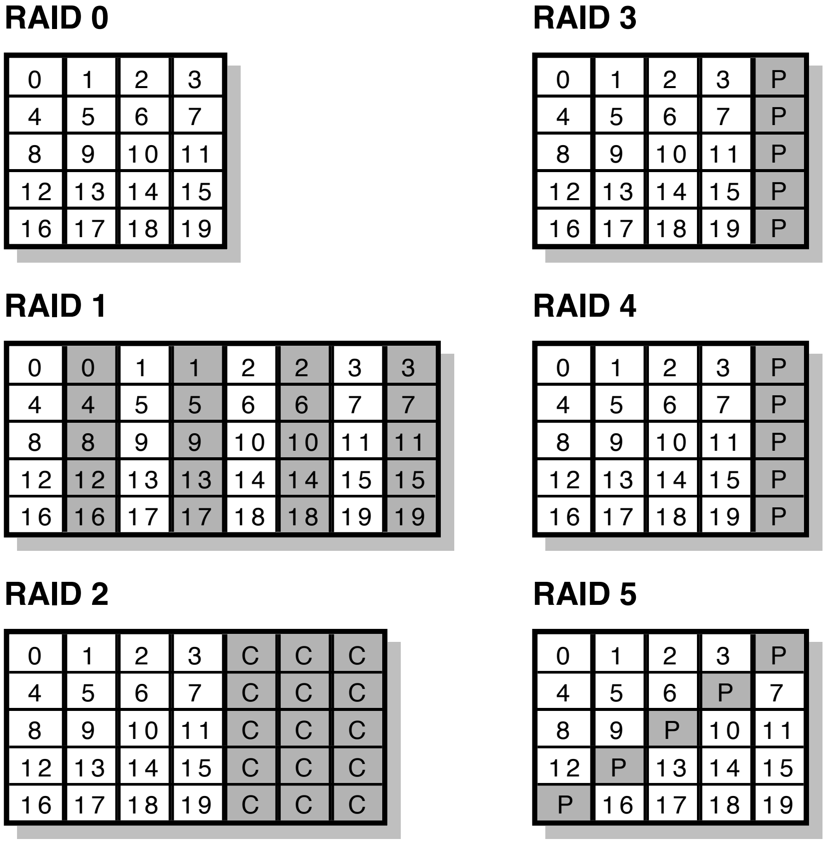
RAID Level 0. Simple disk striping with no redundancy.

RAID Level 1. Otherwise known as *disk mirroring.* Disks are paired, and every write is sent to both disks. If a disk fails, its mirror can be used instead.

RAID Level 2. Hamming code. Data is striped across N data disks. Compute a Hamming code [36] for each group of N bits, one taken from each data disk at corresponding positions, to produce a larger set of bits. Add several "check" disks, so that you can distribute the coded bits one per disk. Since a Hamming code is designed to detect and correct errors, the bit lost due to a disk failure can be recovered using the extra Hamming-code bits stored on the check disks. For $N = 10$ disks, 4 check disks are required; for $N = 25$ disks, 5 check disks are required. Thus, fewer disks are required than in RAID level 1. The Thinking Machines DataVault [64] was one successful RAID 2 product.

RAID Level 3. Single-bit parity. Since, when a disk fails, it is known to have failed, and the identity of the failed disk is known, a single *parity bit* for each N-bit data word is sufficient to reproduce the lost bit in that word. Thus, RAID level 3 uses only one "parity disk" for any group of size N.

RAID Level 4. Block-sized striping unit. RAID level 3 is effective for large reads and writes, each of which span all of the disks. Some workloads, such as transaction processing, tend to make smaller read and write requests. RAID level 4 uses blocks instead of bits as the striping unit, although parity is computed in the same way: one parity bit is produced from N bits, one from each disk at corresponding positions. Thus, it is possible to concurrently read different blocks of data from each data drive, unlike in RAID 3.



0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15
16	17	18	19

0	0	1	1	2	2	3	3
4	4	5	5	6	6	7	7
8	8	9	9	10	10	11	11
12	12	13	13	14	14	15	15
16	16	17	17	18	18	19	19

0	1	2	3	C	C	C
4	5	6	7	C	C	C
8	9	10	11	C	C	C
12	13	14	15	C	C	C
16	17	18	19	C	C	C

0	1	2	3	P
4	5	6	7	P
8	9	10	11	P
12	13	14	15	P
16	17	18	19	P

0	1	2	3	P
4	5	6	7	P
8	9	10	11	P
12	13	14	15	P
16	17	18	19	P

0	1	2	3	P
4	5	6	P	7
8	9	P	10	11
12	P	13	14	15
P	16	17	18	19

Figure 7 RAID levels 0 through 5. Each column represents a disk; all examples have the equivalent of 4 data disks. Each row represents one stripe, with one striping unit per disk. The striping unit is typically one bit for RAID 2 and RAID 3, one block for others. Shaded striping units represent redundant information (C for check bits, P for parity bits, or number for copy).

RAID Level 5. Rotated parity blocks. Notice that in a workload of small reads and writes, RAID level 4 requires four one-block I/Os to write a single data block: read the old data and parity blocks, compute the new parity block, and write the new data and parity blocks. Although the data reads and writes are spread over N disks, the parity disk is used for every write request, and thus becomes a bottleneck. RAID level 5 solves this problem by distributing parity blocks across all disks; each stripe still contains N data blocks and one parity block, but their positions are different on each stripe.

The most common RAIDs in use are RAID level 0 (when reliability is not an issue), RAID level 1 (primarily in critical database applications), RAID level 3 (for high-bandwidth large-read and -write applications), and RAID level 5 (for applications with small I/O requests).

There are numerous RAID implementations from many vendors, some implemented in software (in the file system or device driver), and some implemented in hardware and firmware (in the disk controller). There are a few software-RAID systems that distribute data around a network [37, 48, 62]. These systems are intended to support traditional distributed-workstation workloads.

One group at Hewlett-Packard has extensively examined the question of parallel RAID management, beginning with DataMesh [68] and later TickerTAIP [11]. Although these systems were designed primarily for uniprocessors, they do have the potential to be connected to multiple independent processors. In their most recent work they show how to use a hierarchy of RAID level 1 and level 5 to construct an easy-to-use, cost-effective, high-performance disk array [69].

4.2 Connection

An interconnection network is necessary to move data between multiple I/O devices (or I/O nodes) and multiple memories. There are three fundamental issues involved in connecting I/O devices to computational nodes:

- *Is there a separate network, or subnetwork, dedicated to I/O traffic? Or does all I/O traffic share the interprocessor communication network?*

  One extreme is to connect the I/O nodes, or even I/O-device adapters, directly to the primary interconnection network. Another extreme is to provide an entirely separate I/O network, to which each processor is connected. Or, a compromise is to connect each I/O node to a few points in the main network using an "extra" link; most communications between computational nodes and I/O nodes are routed through the main network as well as the link to the I/O node.

  This distinction is important, because I/O-related network traffic often has different characteristics from other interprocessor network traffic. I/O messages tend to be large and bursty, while most other interprocessor messages tend to be smaller. Throughput is usually the goal for I/O-related communication, whereas latency is typically important for other interprocessor messages. Each can cause congestion or contention that negatively impacts the performance of the other [3, 4, 40]. Although a dedicated I/O network can separate the two forms of traffic, it adds cost. Ultimately, the question is whether, for fixed cost, it is better to use one network or two separate networks with less connectivity or bandwidth. Although there has been some research on this issue (such as [32]), there is as yet no definitive answer.

- *Does the network interface include support for DMA (direct memory access) or shared memory? Does it support user-level access, or are kernel privileges required?*

  These issues are critical because an I/O system depends on an ability to move data. Too many systems have fast interconnection networks that are limited to slow performance by an inefficient network interface. Without DMA, for example, the CPU must use programmed I/O, requiring an interrupt to feed each packet into the network (the IBM SP-1 had this restriction, limiting the performance of its parallel file system [28]). Furthermore, while simple DMA makes a big difference, more sophisticated DMA functionality can be extremely useful. For example, if the DMA unit can gather discontiguous memory chunks into a message, or scatter a message into discontiguous memory chunks, extra memory-memory copies can be avoided. Several parallel file systems have found it advantageous to support discontiguous file accesses [19, 28, 53], for which data-reorganizing DMA support would be helpful.

  Since many parallel file systems are implemented as a user-level library on the compute nodes, and a kernel-level server on the I/O nodes, performance improves if messages can be sent and received through the network interface from user level, without kernel intervention, because there is less overhead on the compute nodes. Several research projects demonstrate the benefits of user-level network interfaces [8, 67].

  Shared-address-space systems, by definition, have specialized hardware support for load and store, to remote memories if necessary, from user level. I/O activity would make good use of a block-transfer mechanism, which can be viewed as a form of DMA to or from remote memory. The BBN Butterfly had this feature [5].

- *Is the I/O adapter attached directly to the interconnection network, or to an I/O-processor node?*

  Probably the simplest approach to building a parallel I/O system, particularly if the processor nodes are fairly conventional processor-memory cards, is to add an I/O-bus adapter (such as a SCSI-bus adapter) to some of the processor nodes to form I/O nodes. The I/O devices are then attached to those I/O buses. But an alternative used by some systems is to build a custom adapter that connects the I/O bus directly to the primary interconnection network. This design avoids an extra copy through the I/O-node memory, but without a local I/O processor to manage access to the device, management may be more complicated.

Connection in example architectures

DEC 2100: Most disks (or RAIDs) in the AlphaServer would be attached to one or more SCSI buses, which are in turn attached to the PCI bus.

KSR 2: SCSI-bus adapters are connected to I/O nodes. There is no separate I/O network. The network interface supports a specialized shared-memory protocol.

nCUBE/ten: as shown in Figure 3, device controllers are connected to I/O nodes. The I/O nodes are interconnected by a dedicated network, and are connected to selected compute nodes. The network wires are connected directly to the CPU itself, but are not accessible from user level.

CM-5: device controllers are attached to specialized I/O nodes, which are attached to the interconnection network. I/O nodes have special DMA controllers that can scatter data from the buffer RAM, through the network interface, to multiple compute nodes, in a wide variety of patterns. Alternatively, it can gather data from multiple remote nodes into the buffer. This ability to reorganize data is an important component of the their ability to provide a traditional linear-file model, striped across disks in 16-byte striping units, and yet be able to map the data in the file to different application “geometries” of processors and virtual processors. The compute-node network interface is accessible at user level.

Maspar MP-2: device controllers are attached to the I/O controller and I/O RAM through either a VME bus or an optional, proprietary 200 MB/s I/O bus. The I/O RAM connects to the PEs through the global router, which is not dedicated to I/O. User-level access and DMA are moot questions, as all actions are synchronous and controlled by the ACU.

4.3 Management

Input/Output refers to the process of moving data into memory from a peripheral device, or out from memory to a peripheral device (such as disk, tape, or network). In a multiprocessor, there may be many memories (typically one for each processor) and many peripheral devices. A key issue, then, is *management:* what processors manage access to the devices? There are three common solutions, shown in Figure 8, where the management is

A. centralized on one processor

B. distributed among all processors, or

C. distributed among a subset of processors that are dedicated to I/O.

Typically, as shown in Figure 8, the devices are attached to their managing processor.

The centralized approach is common in SIMD systems, where most management is centralized anyway and the programming model is synchronous. In large MIMD systems, however, it represents a serious potential bottleneck, especially when used with an asynchronous programming model.

Few systems choose to distribute management among all processors, preferring to concentrate I/O hardware on a subset of processor nodes that are usually dedicated to I/O activities. The concentration of I/O hardware on I/O nodes has several advantages over full distribution [26]:

- The number of I/O nodes and devices may be chosen independent of the number of computational nodes, allowing more flexible system configuration.
- I/O nodes may be constructed differently, e.g., with a different CPU, more or less memory, specialized DMA hardware, and of course adapters for peripherals and I/O buses.
- Fewer adapters may be needed.
- System packaging may be simpler, since compute nodes may have different physical characteristics than I/O nodes. Each may fit into different types of racks, for example.
- I/O-service activity does not impact application computation by stealing cycles or memory, or causing unexpected interrupts.

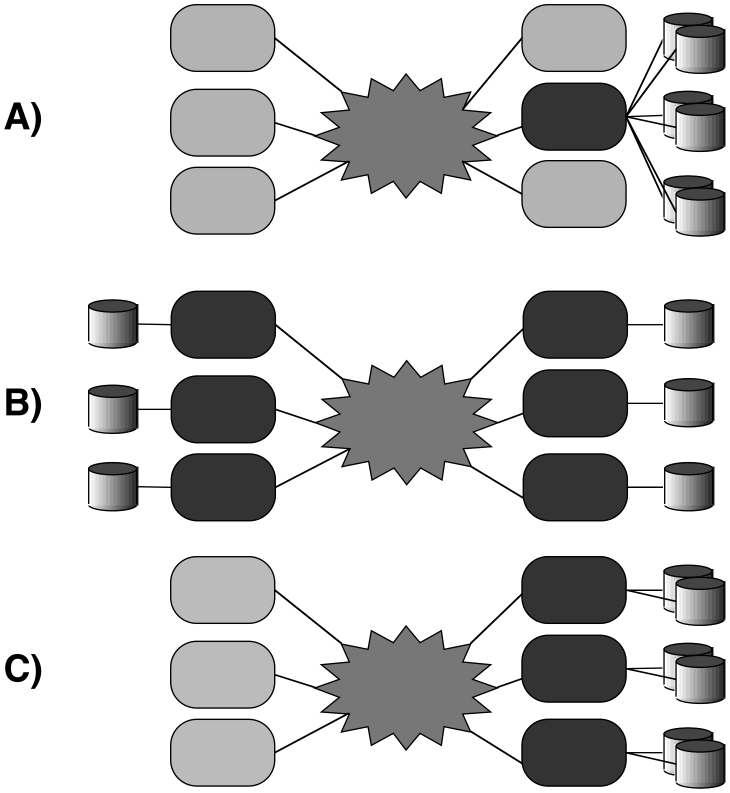


Figure 8 Three common solutions for management of parallel I/O: A) centralized, B) fully distributed, and C) distributed over a dedicated subset.

On the other hand, distributing I/O management among all processors could lead to better locality, if each processor could focus its I/O activity on its local I/O devices. It is difficult to characterize the performance tradeoffs of this locality [43], especially given the wide variety of workloads and interconnection-network architectures, but it seems likely that local disks would be useful for paging and other forms of virtual-memory support for out-of-core computations [15, 17].

Management in example architectures

DEC 2100: Theoretically, it is possible for any processor to manage the devices, although some operating systems may choose to centralize the management on one processor. In a "symmetric" (SMP) operating system, management of all disks is distributed across all processors.

KSR 2: Management and devices are distributed among a subset of processors, though they are not typically dedicated to I/O. Once disk data are read into memory, and that memory is mapped into the application's virtual address space, the shared-memory system handles the movement of data to the appropriate processors.

nCUBE/ten: A dedicated subset of I/O nodes, in conjunction with the host processor, manage all I/O traffic. Compute nodes send requests as messages to I/O nodes.

CM-5: All I/O activity is managed by a partition manager. That is, when a compute node wants I/O, it contacts its partition manager, which then contacts the necessary I/O nodes to arrange a transfer. High-level management is centralized, although there are also dedicated I/O nodes that handle the low-level data flow.

Maspar MP-2: Management is centralized in the Array Control Unit.

Network-attached storage devices

There is an increasing trend to separate device management into high-level and low-level components and to attach the device controller directly to an interconnection network, rather than to a specialized I/O bus. Then a host CPU in one location provides high-level management, while the low-level details are handled by the device controller. This trend is partially a result of the ever-increasing sophistication of device controllers, and by the potential for better performance by moving data directly from the device to the network, bypassing an I/O bus, I/O adapter, and any I/O node's memory. The CM-5 is one specialized example. Other important examples include the RAID-II [24] and HPSS [16, 18] projects. The trend toward network-attached storage devices (NASD) is still new and may have a significant effect on parallel and distributed I/O architecture.

4.4 Placement

All multiprocessors have an interconnection network, and all networks have some topology. Many topologies are more complex than a bus or a ring, such as a hypercube or a mesh. Communication latency, bandwidth, and contention in these networks often depend on the relative position of the endpoints of the communication. Thus, the position of the I/O nodes or devices in the network topology can have a significant impact on the performance of the I/O system. There are three typical approaches:

1. Position is ignored; I/O nodes or devices are placed anywhere.
2. All I/O nodes or devices are clustered in their own "partition" of the network.
3. I/O nodes or devices are distributed around the network, but in carefully chosen positions.

Position is largely irrelevant in some networks, such as buses and many rings. In many of today's networks, the distance between two points is less significant than the message-startup overhead or the length of the message, so position would appear to be unimportant for large-message traffic like I/O. Contention can play a major role, however; if I/O nodes are clustered, traffic to the I/O cluster may be forced through a "narrow" subset of the network. On the other hand, if I/O nodes are distributed around the network, I/O traffic may interfere with other interprocess communications. There have been many studies of this issue, particularly in hypercube networks [30, 32, 58, 70], but also in other networks [3, 4, 40]. Again, there is no commonly accepted solution. Often, packaging issues play a more dominant role in I/O-node placement than do performance issues.

Placement in example architectures

DEC 2100: not an issue, since the topology is flat.

KSR 2: no special placement is necessary or, it seems, suggested.

nCUBE/ten: I/O nodes exist outside of the primary hypercube network, with connections from I/O to compute nodes spaced evenly among compute nodes.

CM-5: I/O processors are clustered together in their own partition. Thus, as I/O traffic goes through the fat tree, it goes "over" rather than through uninvolved partitions (see Figure 4). Inter-I/O-node transfers remain entirely within the I/O partition.

Maspar MP-2: there is only one I/O "node," the I/O controller, and it is connected (through the global router) to all PEs.

4.5 Buffering

Buffering and caching are important aspects of any I/O system. Buffering is important, for example, between a disk drive and an interconnection network, to compensate for the different speeds, different granularity (blocks or packets), and burstiness due to device characteristics (disk seeks) or load (network congestion). A buffer cache, which is an associatively addressed buffer pool holding recently used blocks, is important because it can often avoid I/O entirely. A buffer cache can be particularly important in the I/O node of a multiprocessor, because it can take advantage of *interprocessor locality,* when multiple processors are accessing different parts of the same block [44].

All I/O systems have buffering in several places. We expect to see small speed-matching buffers in the interconnection network, network interfaces, and device adapters. We expect to see buffers and caches inside the disk or tape controllers, and memory caches in CPUs and processor boards. And, of course, operating systems often use some RAM memory for a file-system buffer cache. Of interest here are systems that have explicit buffer or cache hardware set aside for I/O, beyond the usual hardware described above.

Buffering in example architectures

DEC 2100: nothing special.

KSR 2: nothing unusual.

nCUBE/ten: each I/O board has 4 MB, of which 128 KB is dedicated for each I/O node, and the remaining 2 MB is used for the host processor. Some of this memory is used by system software for I/O buffering.

CM-5: each I/O node has 8 MB of RAM dedicated to buffering.

Maspar MP-2: the I/O controller has 8 MB of memory, augmented by up to 1 GB of I/O RAM, all dedicated to I/O. This buffer space is important, to permit data to be rearranged between its layout in the file and its distribution across processors. The file system also manages it as a buffer cache.

4.6 Availability

A multiprocessor system is made up of many components, used in parallel to improve performance. When a file's data are distributed across multiple storage devices, the failure of any device (and subsequently the loss of data stored on that device) effectively causes the loss of the file. Thus, distributing data across multiple storage devices may increase performance, but it *decreases* availability. Disk failure can be masked by redundant disk arrays (RAIDs). I/O-node failure is more complicated; Feitelson et al. [26] describe a clever method to handle this case.

For maximum fault tolerance, failure of other components must also be considered. For example, if all disks in an array are connected to the same controller, power supply, fan, or cable, the failure of any one of those components leads to the failure of the entire array. Thus, some systems provide redundant copies of the components so that the failure of any one component does not cause data loss [69].

Availability in example architectures

DEC 2100: Nothing special; depends on hardware or software RAIDs.

KSR 2: RAID 1 or RAID 3 disk arrays on each I/O node provide security against disk failure. There is no architectural support for RAID across I/O nodes, although the KSR operating system appears to support software RAID across I/O nodes.

nCUBE/ten: There is no specific hardware to support availability. Use of RAIDs would protect against disk failure. It appears that the system can be reconfigured to route messages around failed nodes, although if an I/O node fails, it appears that its controller would be inaccessible.

CM-5: The file system builds a software RAID 3 across disks, with 16-byte striping unit. The architecture includes a special diagnostic network for detecting and diagnosing failures, but otherwise there is no unusual architectural support to increase disk-system availability.

Maspar MP-2: They use a RAID 3 disk array (hardware).

4.7 Database systems

Databases are, of course, I/O-intensive applications. While most of the machines described above can support databases (and many do), there have been several parallel architectures specifically designed to support databases [10, 21, 22, 63]. The Teradata DBC/1012 [63] is perhaps one of the most interesting. In this machine, the processor nodes are arranged in a binary-tree interconnect, with I/O nodes and disk drives at the leaves of the tree, specialized data-merging processors in the internal nodes, and one control processor at the root. This structure is thus designed for the selection, merging, and sorting operations common in database queries. It appears to be specialized for intra-query parallelism rather than inter-query parallelism. Dewitt and Gray discuss parallel database machines in more detail [21].

5 TAPE I/O

Most modern multiprocessors support tape devices, because many multiprocessors are used for data-intensive scientific or commercial applications, and tapes are a cost-effective form of tertiary storage. Most connect standard tape drives through a SCSI- or VME-bus, just like any disk drive. The CM-5 actually has a specialized tape node, which is quite similar to the disk node in Figure 5. A more interesting approach is *tape striping*, in which data from a single file is striped across several tapes in several tape drives, for increased bandwidth [14, 23]. It appears to be difficult to obtain high performance from tape striping unless the workload is primarily large, sequential transfers [23].

6 GRAPHICS I/O

Few multiprocessors have attempted to support parallel graphics hardware, despite the common use of visualization in scientific multiprocessor applications. The nCUBE/2 has the most interesting approach, which allows a single framebuffer to be written in parallel [6, 19]. As with the disk-I/O boards, the nCUBE/2 graphics board has 16 I/O nodes and 128 connections to compute nodes. The I/O-node memory is dual-ported video RAM, which is used as a framebuffer by a high-quality display. Thus, the framebuffer can be modified by sending data to the appropriate I/O node, which then writes it into the appropriate memory location. Striped graphics!

7 NETWORK I/O

Multiprocessors have always supported external networks; the early generation (BBN Butterfly I, Intel iPSC/1, Cosmic Cube, etc.) typically had an Ethernet connection but no local disk drives. Most modern multiprocessors connect to external networks by attaching a network interface to one of the processor nodes. With a fast external network, such as a HIPPI network, it is important to consider how to smooth the flow of data from compute nodes to the I/O node and thence to the external network, or vice versa, especially when the data must be gathered from (or scattered to) many compute nodes [9, 34]. On the other side of the network interface, a industry-government consortium has defined a protocol for parallel data transfers across multiple network connections between distributed supercomputers and network-attached peripherals [7].

The CM-5 has specialized HIPPI-network nodes [66]; they are similar to the disk node in Figure 5 except that they have *eight* interfaces to the CM-5 data network. These eight 20 MB/s connections provide enough connection bandwidth to service the 100 MB/s HIPPI bandwidth.

The nCUBE/2 also supports HIPPI by using multiple internal-network connections to feed one HIPPI network [19]. As with the disk and graphics boards, the HIPPI-network board has 16 I/O nodes and 128 connections to compute nodes. The I/O-node memory is dual-ported video RAM, and shared with the HIPPI DMA hardware. Thus, compute nodes send data to the I/O nodes, who write it into buffers in the RAM. The HIPPI interface reads data out of those buffers and writes it onto the network.

The Maspar MP-2 attaches a HIPPI controller to its I/O bus, much like the disk array in Figure 6 [51]. Again the I/O RAM serves as a buffer between the HIPPI network and the internal global router.

8 SUMMARY

We describe the fundamentals of I/O architecture for multiprocessors, including a review of uniprocessor I/O architecture and disk arrays. Our discussion focuses on disk subsystems, and in particular the following design issues: connection, management, placement, buffering, and availability. We use several machines as recurring examples, including the DEC AlphaServer 2100, KSR 2, nCUBE/ten, CM-5, and Maspar MP-2. We also briefly cover database systems, tapes, external networks, and graphics.

There are other good surveys, although there is no single comprehensive survey of parallel I/O architecture. See [61] for a taxonomy of older disk architectures, chapter 9 of [55] for a good textbook presentation, [41] for a discussion of low-level I/O architecture leading up to a discussion of RAID, [13, 33] for coverage of RAID, and [26, 27, 40] for other excellent overviews of parallel-I/O architecture.

Acknowledgements

Many thanks to Nils Nieuwejaar and Jonathan Howell for their comments on early drafts of this paper, to Mike del Rosario for help understanding nCUBE details, and to Mike Best for help understanding CM-5 details.

REFERENCES

[1] George S. Almasi and Allan Gottlieb. *Highly Parallel Computing*. Benjamin/Cummings, 2nd edition, 1994.

[2] Gene M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *AFIPS Conference Proceedings*, volume 30, pages 483–485, Spring 1967.

[3] Sandra Johnson Baylor, Caroline Benveniste, and Yarsun Hsu. Performance evaluation of a massively parallel I/O subsystem. This volume.

[4] Sandra Johnson Baylor, Caroline B. Benveniste, and Yarson Hsu. Performance evaluation of a parallel I/O architecture. In *Proceedings of the 9th ACM International Conference on Supercomputing*, pages 404–413, Barcelona, July 1995.

[5] BBN Advanced Computers, Cambridge, MA. *Inside the Butterfly Plus*, October 1987.

[6] Robert E. Benner. Parallel graphics algorithms on a 1024-processor hypercube. In *Proceedings of the Fourth Conference on Hypercube Concurrent Computers and Applications*, pages 133–140. Golden Gate Enterprises, Los Altos, CA, 1989.

[7] Lawrence Berdahl. Parallel transport protocol proposal. Lawrence Livermore National Labs, January 3, 1995. Draft.

[8] Matthias A. Blumrich, Kai Li, Richard Alpert, Cezary Dubnicki, Edward W. Felten, and Jonathan Sandberg. Virtual memory mapped network interface for the SHRIMP multicomputer. In *Proceedings of the 21st Annual International Symposium on Computer Architecture*, pages 142–153, 1994.

[9] C. Bornstein and P. Steenkiste. Data reshuffling in support of fast I/O for distributed-memory machines. In *Proceedings of the Third IEEE International Symposium on High Performance Distributed Computing*, pages 227–235, August 1994.

[10] J. C. Browne, A. G. Dale, C. Leung, and R. Jenevein. A parallel multistage 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.

[11] Pei Cao, Swee Boon Lim, Shivakumar Venkataraman, and John Wilkes. The TickerTAIP parallel RAID architecture. *ACM Transactions on Computer Systems*, 12(3):236–269, August 1994.

[12] *Convex Exemplar Scalable Parallel Processing System*. Convex Computer Corporation, 1994. Order number 080-002293-000.

[13] Peter M. Chen, Edward K. Lee, Garth A. Gibson, Randy H. Katz, and David A. Patterson. RAID: high-performance, reliable secondary storage. *ACM Computing Surveys*, 26(2):145–185, June 1994.

[14] Tzi-cker Chiueh. Performance optimization for parallel tape arrays. In *Proceedings of the 9th ACM International Conference on Supercomputing*, pages 375–384, Barcelona, July 1995.

[15] Alok Choudhary, Rajesh Bordawekar, Michael Harry, Rakesh Krishnaiyer, Ravi Ponnusamy, Tarvinder Singh, and Rajeev Thakur. PASSION: parallel and scalable software for input-output. Technical Report SCCS-636, ECE Dept., NPAC and CASE Center, Syracuse University, September 1994.

[16] Samuel S. Coleman and Richard 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*, volume I, pages 5–14, 1993.

[17] Thomas H. Cormen and David Kotz. Integrating theory and practice in parallel file systems. In *Proceedings of the 1993 DAGS/PC Symposium*, pages 64–74, Hanover, NH, June 1993. Dartmouth Institute for Advanced Graduate Studies. Revised as Dartmouth PCS-TR93-188 on 9/20/94.

[18] Robert A. Coyne, Harry Hulen, and Richard Watson. The high performance storage system. In *Proceedings of Supercomputing '93*, pages 83–92, 1993.

[19] Juan Miguel del Rosario. High performance parallel I/O on the nCUBE 2. *Transactions of the Institute of Electronics, Information and Communications Engineers*, J75D-I(8):626–636, August 1992.

[20] 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.

[21] 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.

[22] David J. DeWitt, Shahram Ghandeharizadeh, Donovan A. Schneider, Allan Bricker, Hui-I Hsaio, and Rick Rasmussen. The Gamma database machine project. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):44–62, March 1990.

[23] Ann L. Drapeau and Randy H. Katz. Striping in large tape libraries. In *Proceedings of Supercomputing '93*, pages 378–387, 1993.

[24] Ann L. Drapeau, Ken W. Shirrif, John H. Hartman, Ethan L. Miller, Srinivasan Seshan, Randy H. Katz, Ken Lutz, David A. Patterson, Edward K. Lee, Peter H. Chen, and Garth A. Gibson. RAID-II: a high-bandwidth network file server. In *Proceedings of the 21st Annual International Symposium on Computer Architecture*, pages 234–244, 1994.

[25] Bob Duzett and Ron Buck. An overview of the nCUBE 3 supercomputer. In *Proceedings of the Fourth Symposium on the Frontiers of Massively Parallel Computation*, pages 458–464, 1992.

[26] Dror G. Feitelson, Peter F. Corbett, Sandra Johnson Baylor, and Yarson Hsu. Parallel I/O subsystems in massively parallel supercomputers. *IEEE Parallel and Distributed Technology*, pages 33–47, Fall 1995.

[27] Dror G. Feitelson, Peter F. Corbett, Yarson Hsu, and Jean-Pierre Prost. Parallel I/O systems and interfaces for parallel computers. In *Multiprocessor Systems — Design and Integration*. World Scientific, 1996. To appear.

[28] Dror G. Feitelson, Peter F. Corbett, and Jean-Pierre Prost. Performance of the Vesta parallel file system. In *Proceedings of the Ninth International Parallel Processing Symposium*, pages 150–158, April 1995.

[29] M. J. Flynn. Very high-speed computing systems. *Proceedings of the IEEE*, 54(12):1901–1909, December 1966.

[30] Robert J. Flynn and Haldun Hadimioglu. A distributed hypercube file system. In *Proceedings of the Third Conference on Hypercube Concurrent Computers and Applications*, pages 1375–1381, 1988.

[31] James C. French, Terrence W. Pratt, and Mriganka Das. Performance measurement of the Concurrent File System of the Intel iPSC/2 hypercube. *Journal of Parallel and Distributed Computing*, 17(1–2):115–121, January and February 1993.

[32] 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.

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

[34] Thomas Gross and Peter Steenkiste. Architecture implications of high-speed I/O for distributed-memory computers. In *Proceedings of the 8th ACM International Conference on Supercomputing*, pages 176–185, July 1994.

[35] Haldun Hadimioglu and Robert J. Flynn. The design and analysis of a tightly coupled hypercube file system. In *Proceedings of the Fifth Annual Distributed-Memory Computer Conference*, pages 1405–1410, 1990.

[36] R. W. Hamming. Error detecting and correcting codes. *The Bell System Technical Journal*, XXVI(2):147–160, April 1950.

[37] John H. Hartman and John K. Ousterhout. The Zebra striped network file system. *ACM Transactions on Computer Systems*, 13(3):274–310, August 1995.

[38] John P. Hayes, Trevor N. Mudge, Quentin F. Stout, Stephen Colley, and John Palmer. Architecture of a hypercube supercomputer. In *Proceedings of the 1986 International Conference on Parallel Processing*, pages 653–660, 1986.

[39] W. Daniel Hillis and Lewis W. Tucker. The CM-5 connection machine: A scalable supercomputer. *Communications of the ACM*, 36(11):31–40, November 1993.

[40] David Wayne Jensen. *Disk I/O In High-Performance Computing Systems.* PhD thesis, Univ. Illinois, Urbana-Champagne, 1993.

[41] Randy H. Katz, Garth A. Gibson, and David A. Patterson. Disk system architectures for high performance computing. *Proceedings of the IEEE*, 77(12):1842–1858, December 1989.

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

[43] David Kotz and Ting Cai. Exploring the use of I/O nodes for computation in a MIMD multiprocessor. In *IPPS '95 Workshop on Input/Output in Parallel and Distributed Systems*, pages 78–89, April 1995.

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

[45] Kendall Square Research technical summary. Kendall Square Research, 1992.

[46] Thomas T. Kwan and Daniel A. Reed. Performance of the CM-5 scalable file system. In *Proceedings of the 8th ACM International Conference on Supercomputing*, pages 156–165, July 1994.

[47] Charles E. Leiserson, Zahi S. Abuhamdeh, David C. Douglas, Carl R. Feynman, Mahesh N. Ganmukhi, Jeffrey V. Hill, Daniel Hillis, Bradley C. Kuszmaul, Margaret A. St. Pierre, David S. Wells, Monica C. Wong, Shaw-Wen Yang, and Robert Zak. The network architecture of the connection machine CM-5. In *Proceedings of the Fourth Symposium on Parallel Algorithms and Architectures*, pages 272–285, 1992.

[48] Darrell D. E. Long and Bruce R. Montague. Swift/RAID: A distributed RAID system. *Computing Systems*, 7(3):333–359, Summer 1994.

[49] The design of the MasPar MP-2: A cost effective massively parallel multiprocessor. MasPar Computer Corporation report number MP/P-11.92, 1992.

[50] NCR 3600 product description. Technical Report ST-2119-91, NCR, San Diego, September 1991.

[51] John R. Nickolls. The MasPar scalable Unix I/O system. In *Proceedings of the Eighth International Parallel Processing Symposium*, pages 390–394, Cancun, Mexico, April 1994.

[52] John R. Nickolls and Ernie Rael. Data parallel Unix input/output for a massively parallel processor. Technical Report MP/P-17.93, MasPar Computer Corporation, 1993.

[53] Nils Nieuwejaar and David Kotz. Performance of the Galley parallel file system. In *Fourth Workshop on Input/Output in Parallel and Distributed Systems*, May 1996.

[54] 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, June 1988.

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

[56] Paul Pierce. A concurrent file system for a highly parallel mass storage system. In *Proceedings of the Fourth Conference on Hypercube Concurrent Computers and Applications*, pages 155–160. Golden Gate Enterprises, Los Altos, CA, March 1989.

[57] Terrence W. Pratt, James C. French, Phillip M. Dickens, and Stanley A. Janet, Jr. A comparison of the architecture and performance of two parallel file systems. In *Proceedings of the Fourth Conference on Hypercube Concurrent Computers and Applications*, pages 161–166. Golden Gate Enterprises, Los Altos, CA, 1989.

[58] A. L. Reddy, P. Banerjee, and Santosh G. Abraham. I/O embedding in hypercubes. In *Proceedings of the 1988 International Conference on Parallel Processing*, volume 1, pages 331–338, 1988.

[59] Andrew P. Russo. The AlphaServer 2100 I/O subsystem. *Digital Technical Journal*, 6(3):20–28, Summer 1994.

[60] Kenneth Salem and Hector Garcia-Molina. Disk striping. In *Proceedings of the IEEE 1986 Conference on Data Engineering*, pages 336–342, 1986.

[61] Mark Smotherman. A sequencing-based taxonomy of I/O systems and review of historical machines. *Computer Architecture News*, 17(5):5–15, September 1989.

[62] 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.

[63] DBC/1012. Teradata Corporation Booklet, 1988.

[64] Thinking Machines Corporation. *Programming the CM I/O System*, November 1990.

[65] *The Connection Machine CM-5 Technical Summary.* Thinking Machines Corporation, October 1991.

[66] The CM-5 I/O system. Thinking Machines Corporation glossy, 1993.

[67] Thorsten von Eicken, Anindya Basu, Vineet Buch, and Werner Vogels. U-Net: A user-level network interface for parallel and distributed computing. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, pages 40–53, December 1995.

[68] John Wilkes. DataMesh, house-building, and distributed systems technology. *ACM Operating Systems Review*, 27(2):104–108, April 1993.

[69] John Wilkes, Richard Golding, Carl Staelin, and Tim Sullivan. The HP AutoRAID hierarchical storage system. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, pages 96–108, December 1995.

[70] Andrew Witkowski, Kumar Chandrakumar, and Greg Macchio. Concurrent I/O system for the hypercube multiprocessor. In *Proceedings of the Third Conference on Hypercube Concurrent Computers and Applications*, pages 1398–1407, 1988.