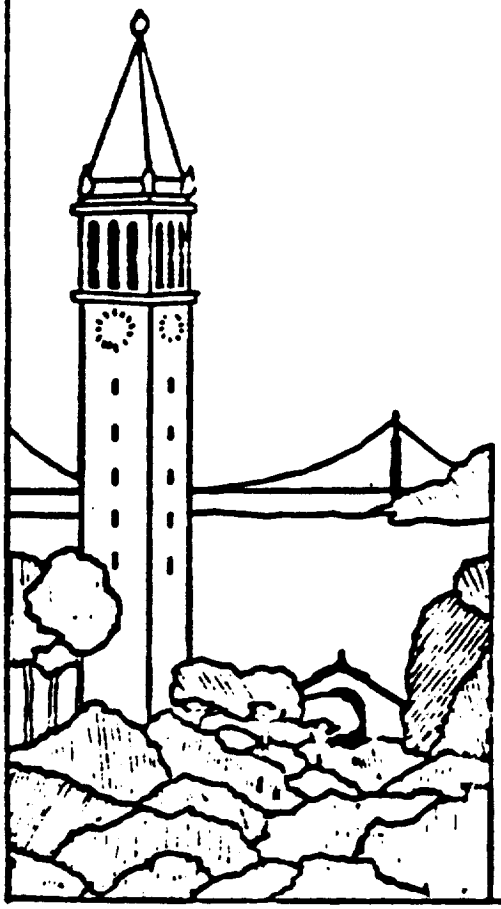
Performance Measurements of the First RAID Prototype

Ann L. Chervenak



Report No. UCB/CSD 90/574

May 1990

Computer Science Division (EECS)
University of California
Berkeley, California 94720

Performance Measurements of the First RAID Prototype

Ann L. Chervenak

May 17, 1990

Abstract

This paper examines the performance of RAID the First, a prototype disk array built by the RAID group at U.C. Berkeley. A hierarchy of bottlenecks was discovered in the system that limit overall performance. The most serious is the memory system contention on the Sun4/280 host CPU, which limits array bandwidth to 2.3 MBytes/sec. The array performs more successfully on small random operations, achieving nearly 300 I/Os per second before the Sun4/280 becomes CPU-limited. Other bottlenecks in the system are the VME backplane, bandwidth on the disk controller, and overheads associated with the SCSI protocol. All are examined in detail.

The main conclusion of this report is that to achieve the potential bandwidth of arrays, more powerful CPUs alone will not suffice. Just as important are adequate host memory bandwidth and support for high bandwidth on disk controllers. Current disk controllers are more often designed to achieve large numbers of small random operations, rather than high bandwidth. Operating systems also need to change to support high bandwidth from disk arrays. In particular, they should transfer data in larger blocks, and should support asynchronous I/O to improve sequential write performance.

1 Introduction

The increasing performance gap between CPUs and I/O systems threatens to create an I/O bottleneck that will limit overall system performance [RAID]. Disk arrays are one attractive solution to this problem. Enormous I/O bandwidth can be achieved if many disks can be accessed in parallel on an array. RAIDs (Redundant Arrays of Inexpensive Disks) add redundancy to arrays in an attempt to make them at least as reliable as a single, large high-reliability disk.

In 1989, the RAID group at U.C. Berkeley built a prototype RAID to test the different striping and parity schemes proposed in [RAID]. This prototype, called RAID the First (RAID-I), was constructed from commercially available 5 1/4" SCSI disks, disk controllers, and a workstation acting as a host processor.

RAID-I performs fairly successfully on small random operations, achieving nearly 300 I/Os per second before host CPU utilization begins to limit performance. Unfortunately, RAID-I is not as successful on large sequential operations. For this workload, host memory system contention limits system bandwidth to 2.3 MBytes per second. Other bottlenecks in the system are the VME backplane, bandwidth on the disk controller, and overheads associated with the SCSI protocol.

The results of this study suggest that to achieve the potential bandwidth of arrays, more powerful CPUs alone will not suffice. Just as important are adequate host memory bandwidth and support for high bandwidth on disk controllers. Current disk controllers are more often designed to achieve large numbers of small random operations, rather than high bandwidth. Operating systems also need to change to support high bandwidth from disk arrays. In particular, they should transfer data in larger blocks, and should support asynchronous I/O to improve sequential write performance.

This paper describes the RAID-I prototype. It presents measurements of the performance this configuration has delivered, and explains the performance limitations present in different parts of the system. In Section 2, the system hardware is described in detail and the SCSI protocol is discussed. In Section 3, the methods and tools used for the research are discussed. Section 4 examines the performance of the system, describes performance bottlenecks, and attributes overheads to various pieces of the hardware. These results include measurements for I/O performed on raw disk devices, as well as those that include the software overhead of the Sprite operating system and of RAID software. Finally, Section 5 draws conclusions about RAID-I performance, and discusses plans for a second RAID prototype.

2 RAID the First

This section presents a description of the hardware that makes up the first RAID prototype. It also touches briefly on some software issues that will be important in subsequent sections of the report.

Figure 1 shows the configuration of RAID-I. The components of RAID-I are off-the-shelf parts that use standard interfaces. There is a Sun4/280 host processor that is attached over a VME backplane to Interphase Jaguar Host Bus Adaptors. The Jaguars control strings of Imprimis Wren IV disks. The Jaguars and Wrens communicate using the SCSI protocol.

We chose the SCSI interface to the disks for several reasons. SCSI is a widely available industry standard. SCSI disks are available in very small formats, and are very inexpensive because their price is driven by the PC market. In addition, SCSI implementations are highly integrated and

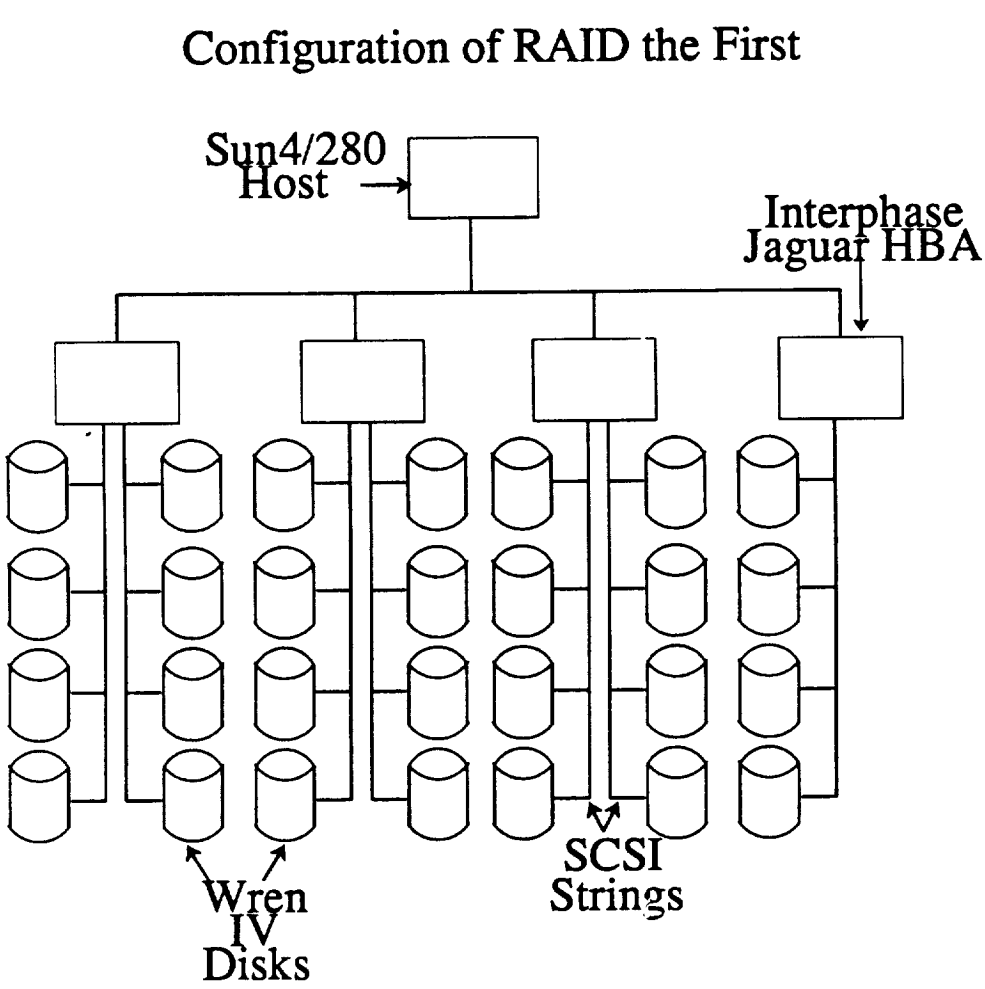


Figure 1: Hardware configuration of RAID-I.

intelligent, which was appealing to us in our desire to use components that were as independent as possible. Finally, Imprimis donated 32 Wren IV SCSI disks for use in RAID-I. Because SCSI is an intelligent protocol, its implementation incurs more overhead than simple protocols. We chose to use it in spite of this performance penalty because its intelligence made our work easier. The measured overheads of the SCSI components are discussed in Section 4.6.

The RAID-I host processor is a Sun4/280, a 32-bit RISC machine that uses a SPARC chip capable of about 10 VAX MIPS. Devices attached to the Sun4/280 transfer data to and from it across the VME backplane, through the Sun4's virtual memory. (Limitations of the Sun4/280's direct virtual memory addressing scheme are discussed below in Section 2.4.2.) The Sun4/280 host processor is named "raid", and it runs the Sprite operating system, developed at U.C. Berkeley.

Attached to the host processor are several Host Bus Adaptor (HBA) boards, which serve as interfaces between the host and disks attached to SCSI strings. The HBAs used in RAID-I are Interphase Jaguar HBAs [Jaguar]. Each Jaguar board can control the communication for two SCSI strings, with four disks per string. In all, there are 7 SCSI strings currently in the array, attached to four Jaguar HBA boards, for a total of 28 disks in the system.

Software on the host allows the Sun4/280 to communicate with the Jaguar. To access individual disks, Mendel Rosenblum, a member of the Sprite group, wrote a Jaguar-specific SCSI device driver that changes commands to the Jaguar format and sends them to the HBA. To access groups of disks in one of the various RAID levels, Ed Lee, a member of the RAID group, wrote a RAID driver that controls striping, parity and reconstruction [Lee].

The remainder of this section focuses on specific pieces of RAID-I. Section 2.1 examines the WrenIV disk, including the use of the Zone Bit Recording technique, buffers on the disk, and the user-settable parameters Buffer Full Ratio and Buffer Empty Ratio. Section 2.2 discusses the setup of commands on the Jaguar Host Bus Adaptor. Section 2.3 describes the SCSI protocol used by the Jaguar and the Wren IV disks. Finally, some problems encountered with the Sprite and the Sun4/280 that will be important in the subsequent discussion on performance are included in Section 2.4.

2.1 The WREN IV Disk

The Imprimis Wren IV is a 5 1/4", 344 MByte disk drive. The disks used in RAID-I have an 80188 processor and a SCSI interface using the Emulex chip. Characteristics of the Wren IV disks are summarized in Table 1 [Wren].

2.1.1 Zone Bit Recording on the Wren IV

The Wren IVs make use of the technique of Zone Bit Recording, resulting in a variable number of sectors per track [ZBR]. Unlike most disks, in which each track on the disk has a fixed number of sectors determined by the length of the innermost track, the Wrens make use of the added capacity available from track lengths that are progressively larger as the disk head moves from the innermost to the outermost track. Adjacent tracks differ in capacity by only a few bytes. Since sectors never span track boundaries on a Wren, extra bytes available on successive tracks are ignored until there are enough unused bytes on a given track to form the largest allowed sector (4096 bytes) plus at least 100 bytes for overhead information. Once this amount of capacity has accumulated, a new sector is made available on this track and on all subsequent tracks of larger diameter. The disk is thus divided into "zones", with all tracks in a zone having the same number of available sectors.

Because there are different numbers of sectors per track, the transfer rate of data off the disk head is also variable. It is highest at the outermost track, averaging a sustained rate of 1.3 MBytes/sec.

2.1.2 Disk Buffers

Although the data comes off the disk head at this rate of approximately 1.3 MBytes/sec, it is transferred between the disk and the host across the SCSI bus at 4 MBytes/sec, a rate that is negotiated between the disk and the HBA when power is turned on. To make such transfers possible, the Wren IV is equipped with a 32KByte disk buffer. Data sent across the SCSI bus goes directly to or from the disk buffer, rather than the disk medium. The disk buffers can be used as simple speed-matching buffers, or they can perform read-ahead in order to improve the performance of sequential read operations. The effect of enabling the disk buffer for read-ahead is examined in Section 4.2.1.

2.1.3 Buffer Full Ratio and Buffer Empty Ratio Parameters

Two user-settable parameters, the Buffer Full and Buffer Empty Ratios, control the operation of the disk buffers. On a read operation, the disk waits until as much data as is mandated by the Buffer Full Ratio (BFR) parameter has been transferred off the disk and into the disk buffer before initiating a reconnect to transfer the data to the host. On a write operation, data is transferred from the host across the SCSI bus to the disk buffer; when the buffer fills, the disk disconnects from the SCSI bus to free the bus while the disk empties some of the data from its disk buffer onto the disk medium. The disk will not reestablish communication with the host (reconnect) to receive more data into its disk buffer until the disk consumes (empties) enough data from its buffer to satisfy the Buffer Empty Ratio (BER). (See Section 2.3 for a complete description of the SCSI protocol, including the use of disconnects and reconnects.)

In the SCSI protocol, the BER and BFR are expressed as numbers from 0 to 255, and the ratios are obtained by dividing those numbers by 255. The resulting ratio is multiplied by the size of the disk buffer on a particular SCSI disk to obtain the number of bytes represented by the BFR and BER for a specific disk. For example, on the Wrens, a BFR of 80 hex (128 decimal) represents a BFR of 128/255, or 1/2. On the Wrens, this corresponds to 16 KBytes of the 32 KByte track buffer. So, on large reads, the disk will wait until 16 KBytes of read data are in its buffer before attempting to obtain control of the SCSI bus in order to send data across the bus to the host.

Two things should be noted in connection with the BFR and BER. First, I/O completions do not wait for the ratios to be met. For example, if the BFR is 16KBytes, as in the previous example, but the read request is for 4KBytes, as soon as the request is completely within the disk buffer, the reconnect with the host will be initiated. Second, a BFR or BER specifying less than a sector of disk buffer space is not allowed.

Section 4.2.2 examines the effect of various BFR and BER settings on performance.

2.2 The Interphase Jaguar HBA

A Host Bus Adaptor is a disk controller linking a host processor to disks and other peripheral devices attached to SCSI strings.

In RAID-I, the Interphase Jaguar [Jaguar] HBA communicates with the Sun4/280 host via a VME interface. All commands, data and control information passed between the HBA and the host are written into 2 KBytes of shared short I/O space of the VME bus.

To issue a command to the Jaguar, the Jaguar-specific device driver on the host processor sets up an *Input/Output Parameter Block* (IOPB) in the VME short I/O space. The IOPB contains a pointer to the SCSI command that will be sent to the disk, pointers to the data buffer and the status block for the command in the host memory, and other control information.

There are two ways that IOPBs can be submitted to the Jaguar, through the *Master Command Entry* and through the *Command Queue*. On power-up, the Jaguar has a Master Command Entry (MCE) space allocated to it in short I/O space. The MCE contains control information and a pointer to space for an IOPB. Commands entered into the MCE are always executed with the highest priority in the system, and can be used to perform error checking and flushing of the queues. Also, upon power-up, the MCE can be used to initialize a Command Queue and *Work Queues*.

The Command Queue is a circular queue that can be initialized to contain up to 40 slots. Each slot, or *Command Queue Entry* (CQE), contains control information (including the work queue into which the command will be entered) and a pointer to an IOPB. After the initialization of the queues is complete, commands submitted to the Jaguar are sent to the command queue first, unless they go through the MCE interface. Then the controller processor submits requests to the appropriate work queues.

Work queues contain commands destined for a particular device. One Jaguar can support up to 14 Work Queues (one for each device on the two SCSI strings). A Work Queue entry contains the control information from the CQE and a copy of the IOPB referred to in the CQE. Once a command enters a work queue, its CQE on the command queue is freed. From the Work Queues, commands are submitted directly to the disks using the SCSI protocol.

When a command is complete, a command response block (CRB) is constructed in the short I/O space shared by the host and controller, and the host processor is interrupted. The CRB contains status information (work queue number, command tag number, completion status) and a copy of the IOPB (so that the host can match the completed command with the appropriate outstanding one). After the host reads the CRB, the space it occupied in the short I/O space is freed.

One important feature of the Jaguar is that data passed through the Jaguar goes through internal Jaguar buffers. There are 128KBytes of memory allocated on the Jaguar for data buffers and for other data structures. The buffer space is allocated 8KBytes at a time, adding overheads to large operations, as discussed in Section 4.6. The performance of the Jaguar is also discussed in Section 4.3.

2.3 The SCSI Protocol

This section describes the SCSI protocol, used as the interface between the Interphase Jaguar HBAs and the Wren IV disks in RAID-I.

The SCSI (Small Computer System Interface) Protocol allows up to eight devices to communicate on a bus or "string" at sustained speeds of 4-5 MBytes/sec [SCSI]. Future SCSI performance will reach 10-20 MB/sec on fast/wide SCSI. Each device on the string can attain control of the bus as a master or "initiator", or may be designated as a slave or "target" for the operation. In

RAID-I, the host bus adaptors function as initiators at all times, and the disks attached to each SCSI string are always targets that receive commands to perform various I/O operations.

The SCSI protocol consists of a series of "phases", during which specific actions are taken by the controller and the SCSI disks. Because it is a high-level, message-based protocol, SCSI inherently has more overhead than simpler protocols. We chose SCSI partly because of its intelligence; the amount of overhead that we incurred as a result is described in Section 4.6.

Section 2.3.1 describes each phase of the SCSI protocol, and Section 2.3.2 details the protocol, applying it specifically to reads and writes using the Wren IV disk and the Interphase Jaguar HBA.

2.3.1 SCSI Phases

Bus Free: No device is currently accessing the bus.

Arbitration: When the SCSI bus goes free, multiple devices may request (arbitrate for) the bus. Each device on the bus is given a unique string address. During arbitration, priority is given to the SCSI device with the highest address.

Selection: After arbitration by an **initiator** succeeds, the selection phase informs the target that it will participate in the operation. (This phase is followed by the Command phase.)

Reselection: After arbitration by a **target** device succeeds, the reselection phase is entered. This occurs when a target wishes to resume an operation interrupted by a disconnect. Reselection informs the HBA which outstanding operation is about to resume.

Command: During this phase, the initiator (controller) reads the SCSI command bytes from host memory and sends them to the target device over the SCSI bus.

Data Transfer: The two data transfer phases allow data bytes to be sent across the SCSI bus in either direction between initiator and target. Direction is specified with respect to the initiator.

- DATA IN: Data is read from the target and sent to the initiator.
- DATA OUT: Write data is sent from the initiator to the target.

Message Phase: The two message phases allow messages to be sent across the SCSI bus in either direction between initiator and target. Direction is specified with respect to the initiator. These messages are used to perform control functions.

- MESSAGE IN:
  - IDENTIFY: This message from the target after reselection specifies which target device is going to resume its operation.
  - SAVE DATA POINTER: This message from the target before a disconnection tells the controller to save its "place" in the data transfer operation so that the data operation can later resume without retransmitting data bytes.
  - RESTORE DATA POINTER: This message from the target after reconnection tells the host to set the value of the Current Data Pointer to the value of the Saved Data Pointer.

- DISCONNECT: This message from the target indicates that the target is going to give up the SCSI bus in order to perform certain operations internally, for example, to perform a seek, to fill its buffers as specified by the BFR on a read operation, or to empty them as specified by the BER on a write operation.
- COMMAND COMPLETE: This message is sent by the target to inform the initiator that an operation is complete.

• MESSAGE OUT:

- IDENTIFY: This message from the controller at the start of an operation tells the devices on the string the identity of the initiator and the desired target.

Status Phase: Immediately before the COMMAND COMPLETE message is sent, the target sends the initiator status information about the command.

2.3.2 The Protocol

Figure 2 shows the SCSI phase transitions for a typical read operation.

Multiple I/O operations can be in progress at a time through a host bus adaptor. The setup on the host machine for each command requires that the host create a command descriptor block in host memory and allocate space for the operation's data and status blocks. The host then issues the command to the HBA, including pointers to the data, status and command blocks in host memory. The instruction also includes the SCSI address of the target device.

The HBA maintains two sets of pointers for each outstanding operation–the Current and the Saved pointers. (There is at most one outstanding operation per disk.) The Current pointers point to the next byte of command, data or status information to be transferred from host memory to the device. The Saved command and status pointers always point to the start of their respective blocks. The Saved data pointer initially points to the start of the data block. If a disconnect occurs during data transfer, it is preceded by the SAVE DATA POINTER message issued from the target device. This message updates the Saved data pointer to reflect the current position within the data transfer. An interrupted operation may later resume (following a reconnection and a RESTORE DATA POINTER message) from the position in the data transfer where the data pointer was last saved. This eliminates the need to retransmit data bytes.

After the HBA has set up these pointers, it arbitrates for the SCSI bus. When it wins, it selects the SCSI target device. During the selection, it asserts the ATTENTION line. This is a signal to the target that the initiator wishes to send a message. This is necessary, since in the SCSI protocol, all bytes of data and messages sent across the SCSI bus are requested by the target device and acknowledged by the initiator device, regardless of the direction of the transfer.

The disk responds to the ATTENTION (ATN) signal by asserting the REQUEST (REQ) line for the initiator, which responds with an IDENTIFY message in the MESSAGE OUT phase and an assertion of the ACKNOWLEDGEMENT (ACK) line. (The REQ/ACK protocol for all that follows is analogous, and will not be repeated.) The IDENTIFY message contains the target logical unit number (LUN), and indicates whether the initiator is capable of disconnection. (In RAID-I, all devices support disconnection.)

The next phase is the COMMAND phase, in which the target requests successive bytes of the command. In general, the HBA uses DMA to read the command bytes from host memory. In the specific case of the Jaguar HBA, the command bytes at this time are in Jaguar memory space in

SCSI Phase Transitions -- Read Operation

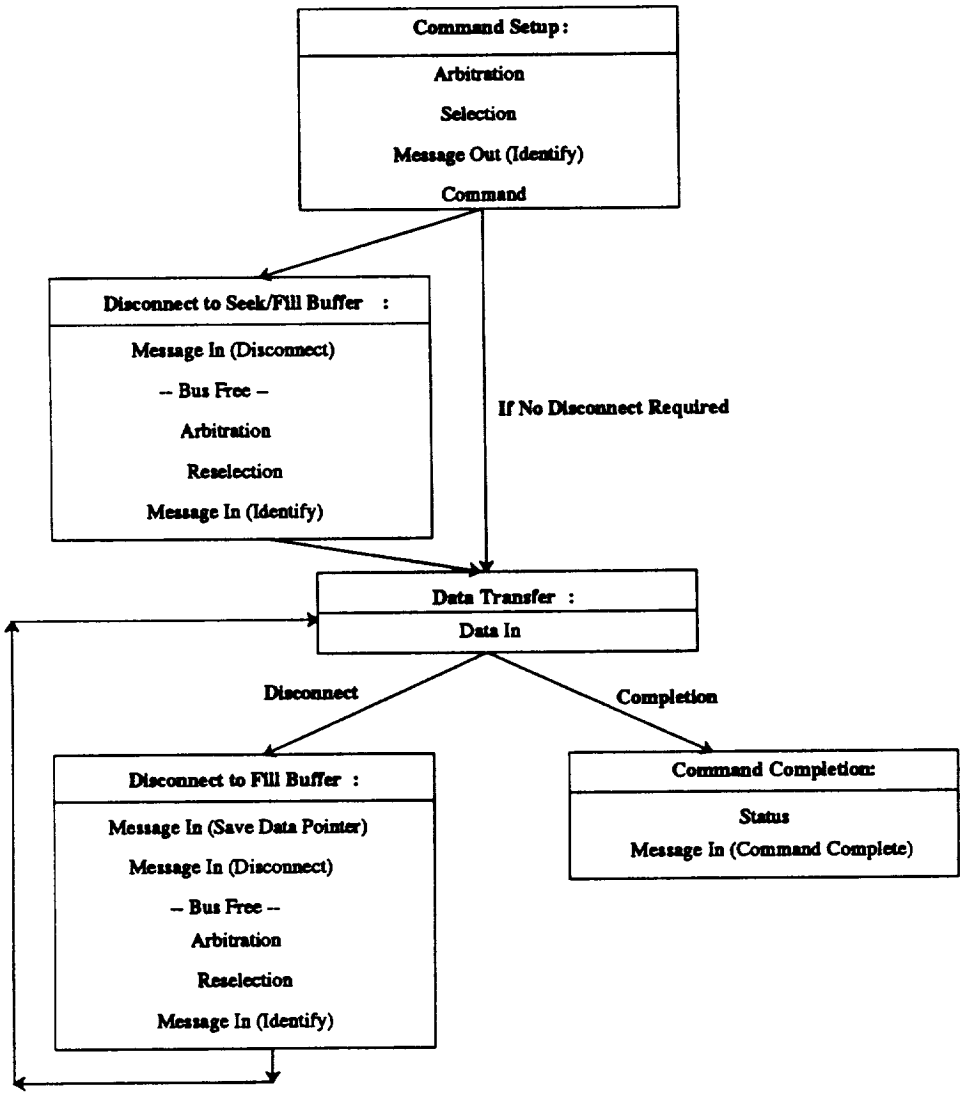


Figure 2: SCSI phase transitions for read operations.

a device-specific Work Queue. From the work queue, the commands are sent across the SCSI bus to the disk.

After it has received the command, the SCSI chip on the target disk decodes the instruction and determines whether a disconnection is necessary. The disk will disconnect if a seek is necessary. Also, on a read operation, a disconnect will occur if the amount of data in the disk's buffer ready to be transferred is less than the entire request, or less than specified by the buffer full ratio (whichever is smaller). On a write, the Wren will accept data into the disk buffer while the seek, if necessary, is being issued; a disconnect will occur when the buffer fills, or when the data for the operation is completely contained in the buffer.

If a disconnect is necessary, the target issues a DISCONNECT message in the MESSAGE IN phase. (Since no data has yet been transferred, the DISCONNECT message is not preceded by a SAVE DATA POINTER message.) The target releases the busy signal and the bus goes free.

When the seek is complete, or when the relevant buffer ratio is met so that the disk is ready to continue the operation, the target device arbitrates for the SCSI bus. When the disk gets the bus, the RESELECTION phase is entered. The disk sends an IDENTIFY message in the MESSAGE IN phase, followed by the RESTORE DATA POINTER message. The HBA then reconstructs the logical thread between the request being resumed and the pointers relating to that operation in the HBA's local memory.

The DATA TRANSFER phase now begins. If the operation is a read, then the phase is the DATA IN phase, and the target device asserts the REQ line as it puts each byte on the SCSI bus. If the operation is a write, then during the DATA OUT phase, the disk requests each byte, and the controller responds by putting the data bytes on the bus.

Data transfer continues until the operation completes or until the need arises for a disconnect. On a read, a disconnect will occur if the disk buffer empties (possible, since data enters the buffer from the disk at 1.3 MBytes/sec and leaves over the SCSI bus at 4 MByte/sec) or if a cylinder boundary is crossed, requiring a seek. On a write, a disconnect will occur whenever the disk buffer fills with data transferred from the host; when enough data is written from the buffer to the disk medium that the buffer empty ratio is met, the operation can be resumed.

To disconnect, the disk first sends a SAVE DATA POINTER message in the MESSAGE IN phase, which the HBA acknowledges after it saves the data pointer. Then the disk sends a DISCONNECT message, and after the HBA acknowledges it, the bus goes free.

As before, when the disk is ready to resume the data transfer, it arbitrates for the SCSI bus and reselects the initiator.

When all the data has been transferred, the target device enters the STATUS phase and sends a status message to the HBA indicating any errors that occurred during the operation.

Finally, the MESSAGE IN phase is entered, as the target sends a COMMAND COMPLETE message to the initiator.

2.4 Problems Encountered with Sprite and the Sun4/280

This section discusses some characteristics of Sprite and the Sun4/280 that affect the performance measurements that will be discussed in Section 4. They are included here as background material. Section 2.4.1 describes the instruction sequence of a Sprite file system read; this sequence affects the performance of RAID-I for large sequential transfers. Section 2.4.2 describes problems encountered with the Sun 4/280's direct virtual memory addressing system.

2.4.1 Sprite

The host processor for RAID-I is the Sun4/280, running the Sprite Network Operating System.

Table 2 summarizes the instructions necessary to issue a read from the Sprite file system. These instructions were counted by Mendel Rosenblum.

This instruction sequence is relevant because it affects the performance of RAID-I for large sequential operations.

Traditional file system workloads are characterized by small transfers, performed 4 KBytes or 8 KBytes at a time. [Ouster] showed that files are generally small, and are read and written in their entirety. Sprite running on the RAID-I host works well for this traditional file system workload, but unfortunately, runs into problems for large operations.

All I/O on Sprite is done through the kernel; when the disk read is complete, for instance, the data must be copied from kernel space into the appropriate user address space that initiated the I/O operation. In addition, since I/O is performed through the cache on the Sun4/280 (described in the next section in more detail), I/O operations incur a cache flush. The copy operation executes at about 5 MBytes/sec, and is the largest problem for fast operation on the host. The cache flushes take place at 12 MBytes/sec if the cache line is dirty, and 78 MBytes/sec if the cache line is clean. The effect of the copies and cache flushes is that the Sun4/280's memory system quickly becomes saturated with so much DMA activity occuring. The effect of this bottleneck can be seen in Section 4.1.2 and Section 4.4.

Because large sequential transfers create a bottleneck in the host memory system when Sprite is used, Mendel Rosenblum wrote a system call to eliminate much of the overhead generally associated with Sprite This system call will be referred to throughout this paper as the "no-copy system call". The purpose of this system call was to give us a measure of the raw performance possible on the hardware for large sequential operations. Section 3.2 discusses the system call in more detail.

2.4.2 Sun4/280 DVMA Problem

An idiosyncrasy of RAID-I's host processor, the Sun4/280, caused problems early in the testing phase of RAID-I. It was discovered that there were never more than a few concurrent I/O operations in progress. Members of the Sprite group traced the problem to the DVMA system used by the Sun4/280.

All DMA from VME-based devices to the Sun4/280 pass through the host's cache using virtual addresses. To read the disk as a raw device, the sequence of events is as follows: the read system call is issued, and the file system allocates a buffer for the data from the I/O operation. This kernel buffer is mapped into a 1 MByte space in the address space of the currently executing process (called the current context). During the I/O operation, the data is written to the mapped kernel buffer. After the data transfer is complete, the kernel buffer is unmapped from the current context and the cache is flushed. Data is copied from the kernel buffer to the user space that requested the data, and the kernel buffer is freed.

Using this scheme, all DVMA must be done through a 1 MByte space in the current context. In all, there are 16 contexts on the Sun4/280, but it is this same 1 MByte space that is mapped into each of the separate contexts. They share it as they share a single copy of the kernel mapped into each of their address spaces. So, in all there is 1 MByte of space available for concurrent DVMA operations to use. (Actually, the space is slightly less than 1 MByte.) This space limitation reduces the number of outstanding I/O operations to a few at a time.

The Sprite group was able to circumvent this limitation by changing the DVMA. Instead of going through the 1 MByte space allotted to it in the current context, DMA is now performed through 2 GBytes of an unused address space. This "fix" virtually eliminated the limitation on the number of concurrent operations that could be issued from the host.

3 Tools and Methods

I used a number of different tools and approaches to analyze the performance of RAID-I. Section 3.1 describes traces of activity on the SCSI bus. Section 3.2 discusses programs used to generate and evaluate I/O performance in the array. Section 3.3 describes modifications of the Sprite kernel I used to stress the array and to trace operating system behavior.

3.1 Traces

Traces of activity on the SCSI bus were collected directly using the Ancot SCSI Bus Analyzer [Ancot]. The Ancot records the timestamps of transitions of any of the following control lines on the SCSI bus: BSY (busy), SEL (select), ATN (attention), RST (reset), MSG (message), I/O (data direction on the bus), C/D (control or data). The Ancot's timer has a resolution of 50 nanoseconds.

The Ancot is designed to allow trace data to be uploaded to an IBM PC/AT. Richard Drewes, a member of the RAID group, wrote a program to upload those traces along a serial RS232 line into the serial port of a Sun3 workstation. Traces were stored on disk, and then downloaded back onto the Ancot for later analysis. Also, the encoded traces were processed on a Sun3, using various awk and C programs to collect statistics on the traces.

Examples of this trace analysis are shown in Tables 3, 5, 7 and 9. The collected data includes the percentage of time spent in each phase of the SCSI protocol and the time in each phase normalized per I/O. (These tables are described in detail in Section 4.)

The size of traces recorded on the Ancot is limited by the size of the device's internal memory (32K), and thus reflects I/O activity over a relatively short time–generally a few seconds. Longer periods of I/O activity were monitored using statistics-gathering programs.

3.2 Programs

Peter Chen (another member of the RAID group), Richard Drewes and I wrote user-level programs that generated I/O operations and gathered statistics on I/O activity. The statistics gathered using these programs reflect the performance of the system including the overhead of the Sprite operating system, described in detail in Section 4.1.2. Most of the programs were very simple, designed to issue some number of random or sequential reads or writes, time the execution of the operations, and report on the MBytes/sec or I/Os/sec achieved by the array. Peter Chen's "mult" program was more sophisticated; it could issue requests as distributions of reads and writes of multiple sizes. These features of the mult program were not used in the tests reported here.

3.3 Modified Sprite Kernel

There were two sets of modifications to the Sprite kernel. The first modifications were intended to circumvent some Sprite processing in order to stress the RAID-I hardware. The second set allowed tracing of Sprite activity during I/O operations.

As discussed in Section 2.4.1, the copy operations and cache flushes that occur during DMA transfers to and from the disk cause a bottleneck on the host CPU's memory system for large or sequential operations. To measure the raw performance of the array for large operations, we needed to eliminate some of these DMA operations. Mendel Rosenblum wrote a system call that accepts a buffer containing up to 512 commands and puts them all on the device driver queue at once. This eliminates much of the overhead normally required to issue subsequent commands, since returning to user level between these commands is not required. Also, this system call allocates a single buffer in DVMA space where the results of all the I/O operations are written. Subsequent operations overwrite the data from the last operation; the data written by an I/O operation into DVMA space is never copied to the user's address space, but is simply discarded. This system call allowed us to measure the performance capabilities of the hardware with minimal processing of the data being transferred. Performance measurements obtained using this system call represent an optimistic upper bound on what the hardware can deliver.

To better understand how the overhead limiting RAID performance is divided between the operating system and the controller, we modified a Sprite kernel to record the timing of certain I/O events. Of interest were the time at which a command is submitted to the device driver by the file system, when the device driver signals the Jaguar controller board that a command is awaiting execution, when the Jaguar interrupts the host upon completion of SCSI activity, and when the operating system completes the operation. Results obtained using this kernel are described in Section 4.7.

4 Performance

We had two performance goals in RAID-I. We hoped to use the many available disk arms in the array to demonstrate the ability of disk arrays to function well in environments that typically generate small, random I/Os, such as database applications and traditional file systems. We also hoped to demonstrate the high bandwidth possible using a RAID for large and sequential I/O operations typical of scientific computing applications. The following sections examine both workload types.

Section 4.1 looks at the performance of disks on a single SCSI string, and the performance impact of host software overhead. Section 4.2 examines the effect of varying user-settable disk parameters: disk buffer readahead and the Buffer Full and Empty Ratios. HBA performance is examined in Section 4.3. Overall large sequential performance (measured in MBytes/sec) is discussed in Section 4.4, while small random I/O rates for the array (measured in I/Os/sec)are described in Section 4.5. Overheads attributable to the disk and the HBA are discussed in Section 4.6. Section 4.7 describes the results obtained from the modified Sprite kernel described in Section 3.3, including timing sequences from start to finish for various I/O operations.

4.1 Disk and String Performance

4.1.1 String Performance with Minimal Sprite

The performance results presented in this section are for I/O operations generated using the special no-copy system call described in Section 3.2.

Sequential Reads and the String Bottleneck

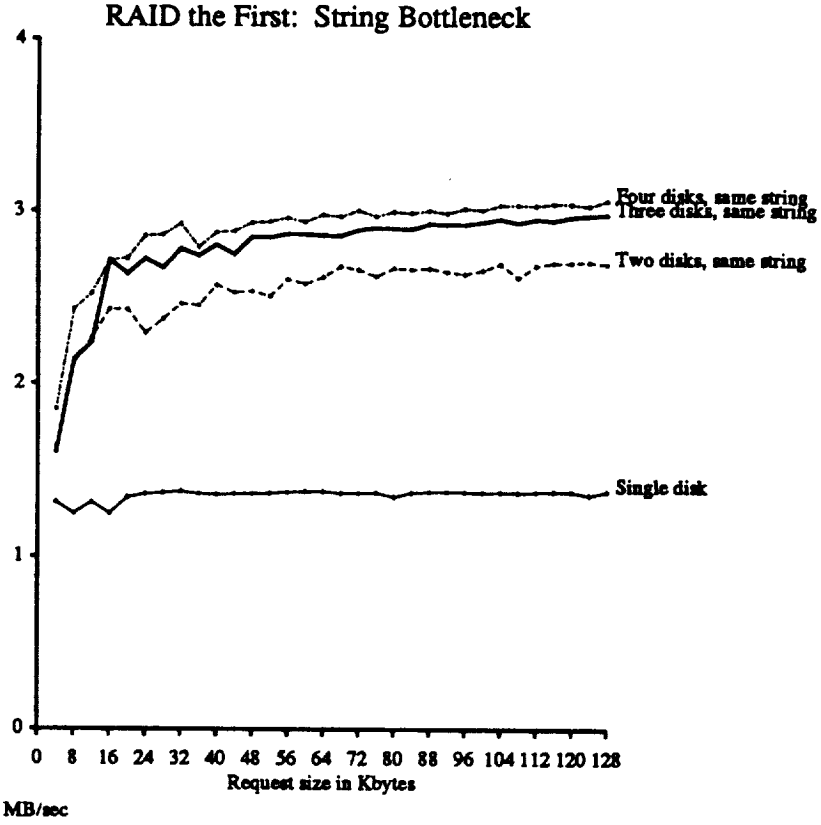


Figure 3: Bandwidth for sequential read operations issued using the no-copy system call to disks on a single string. A separate process issues requests to each active disk; from one to four disks are active at a time.

Figure 3 shows the bandwidth attained on a single SCSI string performing sequential reads using the no-copy system call. The first disk on the string is able to achieve its expected bandwidth of 1.3 MBytes/second. When a second disk is added, the bandwidth approximately doubles for large request sizes. However, when a third and fourth disk are accessed on the string, there is very little performance improvement, indicating the existence of a performance bottleneck. The maximum bandwidth obtained on the string is around 3 MBytes/sec, only 75 % of the expected string bandwidth of 4 MBytes/sec. This 4 MB/sec transfer rate is agreed upon by the Wren IV disk and the Jaguar HBA during a negotiation process in which they participate on power-up. Trace analysis shows that data bytes do pass between the devices at 250 nsec per byte, but that the expected sustained transfer rate of 4 MBytes/sec is not attained because of overheads associated with the disk and controller in implementing the SCSI protocol. These overheads are described in detail in Section 4.6.

Table 3 lists the percentage of time and the normalized time per I/O spent in each phase of the SCSI protocol for a trace of 32 KByte sequential reads. A separate process issued these reads to each of four disks on a single SCSI string.

The time spent in the arbitration phase is very short, indicating that four disks can perform large sequential transfers without consuming much SCSI bandwidth with arbitration overhead. Most of

the other SCSI phases (messages, selection, disconnects, reconnects, etc.) similarly account for very little time in the life of the transaction.

Large sequential transfers are dominated by time spent in the data transfer phase, which accounts for 93% of the time on the SCSI bus during this trace. This is not surprising, since for sequential transfers, there are no seeks between subsequent transactions. Other overheads from the protocol consume only a few milliseconds, and the minimum time in the data transfer phase is 7.8 msec (the time to transfer 32 KBytes over the SCSI bus at 4 MBytes/sec). As mentioned above, there are overheads within the data transfer phase that make the actual transfer time longer than this minimum. In this trace, the time in the data transfer phase is approximately 10 msec per transaction. The overheads that account for this longer transfer time are discussed in detail in Section 4.6.

The SCSI bus is free only 1% of the time in this trace. The bus is fully utilized. Since only 75% of the expected bandwidth from the string is achieved, 25% of the bandwith must be consumed by SCSI overheads. Given these overheads, 3 MBytes/sec is the best bandwidth that can be achieved on a single SCSI string.

Table 3 does not include normalized numbers for the average bus free time during a transaction. Trace analysis does not give an accurate measure of average bus free time per I/O, since in this trace there are four disks active on the SCSI string at a time, and their operations overlap. The time that the bus is actually free in the trace represents the amount of time overall that there is no such overlap.

An interesting question is how busy the host CPU is during this period. Unfortunately, this is not a straightforward question to answer. Idle time on a CPU is measured by counting the number of times that the CPU executes the instructions in an "idle loop". These instructions must be fetched over the memory bus before they can be executed. It is our assertion that when performing intensive I/O, the Sun4/280 host experiences saturation on its memory bus rather than in its CPU because of all the copy operations performed by the operating system. This complicates the CPU utilization measurements, because the DMA for the copy operations has precedence in acquiring the memory bus over the CPU process attempting to execute the idle loop. If there is a lot of traffic over the memory bus, then the CPU will be slow in fetching subsequent instructions of the idle loop, and its idle time will be underestimated in CPU utilization measurements. Thus, it will appear that the CPU is highly utilized when it is actually the host memory system that is highly utilized.

The "CPU utilization" numbers that can be measured, although not accurate measures of the CPU's status for the reason just explained, nonetheless do allow a comparison to be made between different workloads and how they stress the combination of the host memory and CPU. Such numbers will be referred to here as Sun4/280 host utilization measurements. Low host utilization numbers indicate that the Sun4/280's memory system is not highly utilized, and that the CPU has no trouble fetching the instructions to execute the idle loop. High host utilization numbers indicate that either the CPU or the memory system has become a bottleneck. In our case, the memory system is generally the limiting factor for performance, due to the number of copy operations being performed (see Section 2.4.1).

For 32 KByte sequential reads issued by the special no-copy system call to four disks on a string, the host utilization measurement is 7.69%, indicating that the host memory system and CPU are not a performance limitation in this case.

Figure 3 represents performance of sequential reads when the disk buffers are enabled for reada-

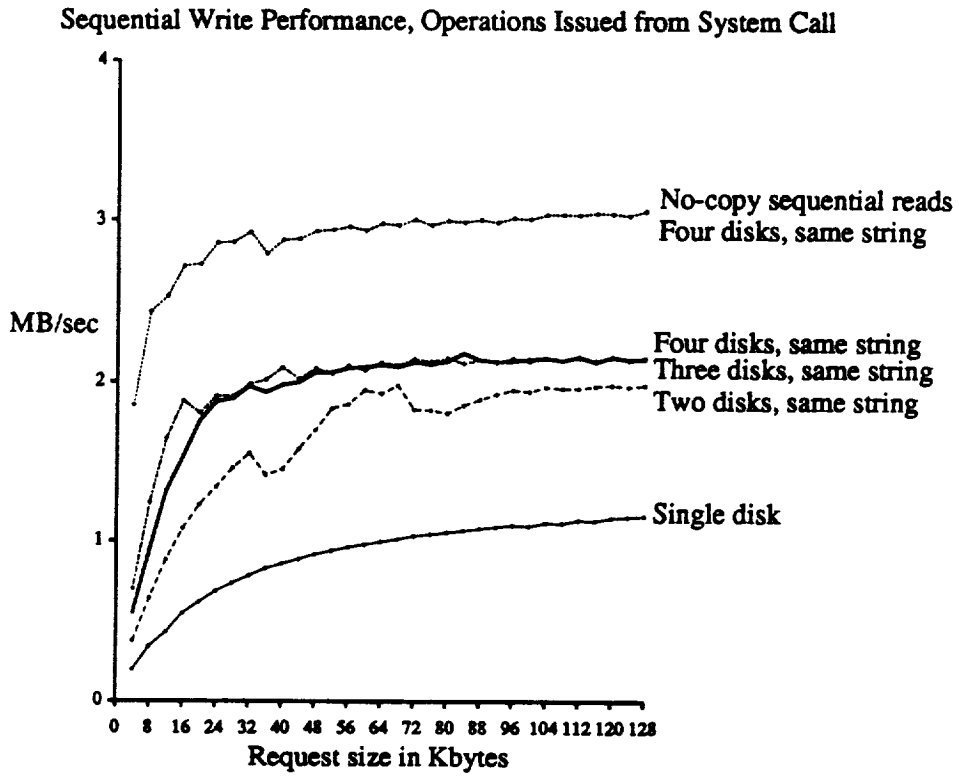


Figure 4: Bandwidth for sequential writes generated using the no-copy system call. A separate process issues requests to each active disk. The top line in the graph is the bandwidth for no-copy sequential reads issued to four disks, included for comparison.

head. Much (or all, for small operations) of the data for a sequential read is contained in the disk buffer before the operation begins. Section 4.2.1 examines the performance consequences of disabling readahead in the disk buffers.

Sequential Writes

Figure 4 shows the performance for sequential writes issued to one, two, three and four disks from the no-copy system call.

The bandwidth achieved for sequential writes is much lower than that for reads. For a single disk, performance approaches 1.1 MBytes/sec only for the largest request sizes. By comparison, in the sequential read case, 1.3 MBytes/sec was achieved even for small request sizes. Four disks active on a string achieve only around 2 MBytes/sec, compared to 3 MBytes/sec in the sequential read case. (The top line in the graph shows the sequential read performance for four disks, for purposes of comparison.)

The disparity in performance results partly from the inability to use readahead, which helped performance in the sequential read case. It also results partly from a longer data transfer phase on write operations. Table 4 shows the breakdown of SCSI phase times for 32 KByte sequential writes

to four disks. On average, 14 msec are spent in the data transfer phase compared to 10 msec for sequential reads.

We examined a trace of 32 KByte sequential write operations on the Wren IV. On every transaction in the trace, the command was sent to the disk immediately followed by data. Each time, the entire 32 KBytes of data were sent to the track buffer before the disk disconnected. Since the new transaction holds onto the data bus in the Data Transfer Phase until all its data has been transferred to the disk buffer, some transactions that could otherwise use the bus are unable to do so. In addition, the data transfer phase of sequential write operations is longer than that of sequential reads because the write operations suffer pauses during data transfer. These pauses begin after the first 8 KBytes of data have been transferred, occur every 512 bytes thereafter, and last for 100 to 400 msec. The pauses are caused either by the Jaguar or by the Wren IV; I was unable to tell which, because the Ancot does not reveal the signals on the request and acknowledge lines that would have revealed which component was holding up the transfer. These delays within the data transfer phase explain why the bus continues to be fully utilized in this case but lower bandwidth is achieved.

Another intersting observation about sequential write performance is that a single write spends much longer on the Jaguar board than a read does. I suggest that the difference is caused by a missed rotation on the sequential write operation. Tables 24 and 25 show a timeline for non-overlapped sequential reads issued from the no-copy system call to a single disk. These timelines show that approximately 16 msec of extra time is spent on the Jaguar board during write operations. About 4 msec of this time is due to the extra time spent in the data transfer phase in the sequential write operation. The other 12 msec appears to be the results of a missed rotation. Between subsequent operations, some processing in the Jaguar device driver and some setup time on the Jaguar board are required; this processing consumes several hundred microseconds. Then SCSI overheads are incurred. By the time the write operation actually is ready to write data from host memory to the disk, the disk position has moved past the starting point for the write operation on the disk. The 12 msec extra time seen in the write trace compared to the read trace corresponds to about 3/4 of a full rotation.

Host utilization is measured at 5.2% for this case, once again indicating that the Sun4/280 CPU and memory system are not a bottleneck for operations issued by the special system call, since the copy operation and most cache flushes are eliminated.

Random Reads

Figure 5 is a graph of I/O rates for small random read operations issued using the no-copy system call for minimal Sprite processing. Table 5 shows the breakdown of the various SCSI phases for two traces. Both involve 4 KByte random read requests to each of four disks on a single string. The first set of measurements is for requests issued by a separate process for each disk. The second set of measurements is for requests issued by four processes per disk.

The first three disks activated on a string get 30 I/Os/sec each for 4 KByte random operations. When four disks are active on a string, there is a slight performance degradation, with each disk consistently achieving only 29 I/Os/sec. This slight decrease in I/O rate per disk for four disks on a string is not the result of string contention; Table 5 shows that the SCSI bus is free approximately 2/3 of the time. The small degradation is also not the result of contention on the host memory

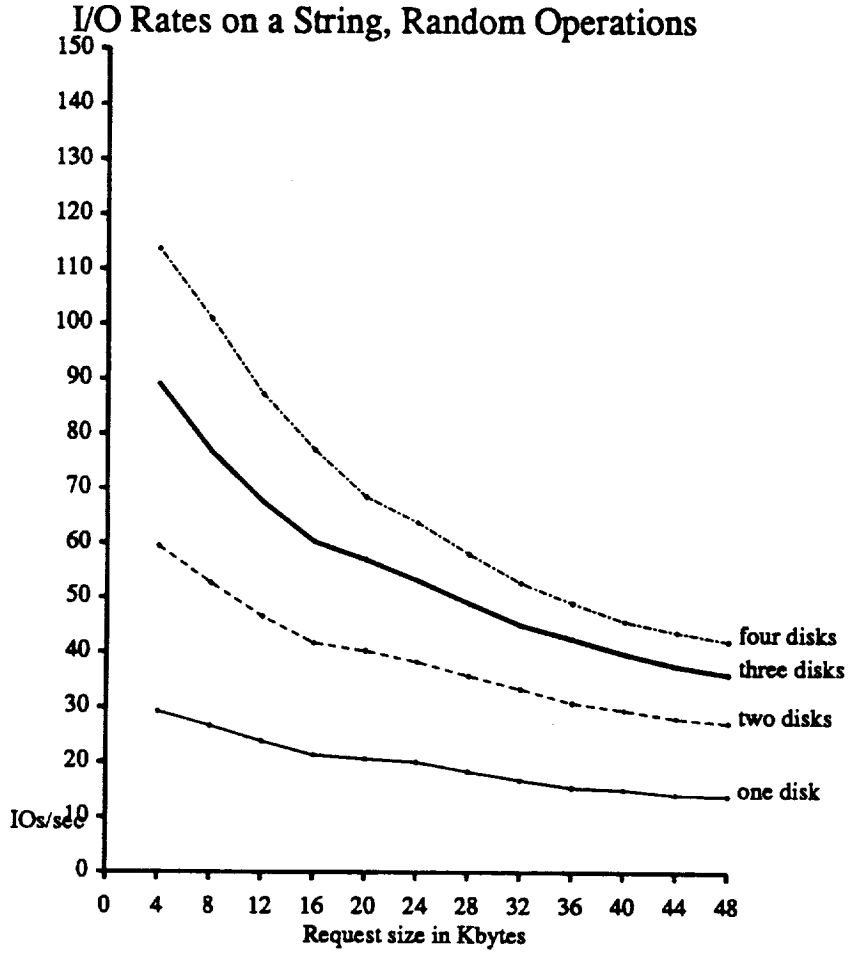


Figure 5: I/O Rates for random operations generated using the no-copy system call. A separate process issues requests to each active disk.

bus or CPU; host utilization measures only 7% in this case. Thus, the slight drop in performance when four disks are active on the string must be caused by the Jaguar's internal processing.

Figure 5 shows that larger random operations suffer significant performance losses when a third and fourth disk on the string are activated. This degradation is caused by utilization of the SCSI string, which measures nearly 80 % for four disks performing 48 KByte operations.

The most dramatic differences between Table 5 and Table 3 are the decreased time spent in the data transfer phase, and the increased percentage of time that the SCSI bus is free. Since the transactions in the trace are small (4 KBytes), a short data transfer phase is expected. The Jaguar controller is optimized for such short transactions, and performs efficiently. Thus, each data transfer phase takes about 1.5 msec, only slightly longer than the minimum transfer time of 1 msec.

The increase in bus free time is also not surprising. These small operations are dominated by average seek and rotation times (26 msec) plus the time to satisfy the Buffer Full Ratio (3.3 msec to move 4 KBytes off the disk head), which dwarf data transfer time (less than 1 msec for 4 KByte transfers) and protocol overheads (a few msec). Even with an operation pending on each of the four disks, the SCSI bus will be idle most of the time, because the disks spend most of their time performing seeks.

The other phases of the SCSI protocol (selection, messages, status, etc.) show some small deviations from Table 3, and because the request size is smaller in this case and the transaction time is shorter, these phases account for a larger percentage of the total than they did in the sequential read case. However, these other phases still represent a small fraction of the lifetime of a transaction.

Random Writes

Small random write performance is actually better than small random read performance, as shown in Figure 6. Each disk on the string achieves about 32 I/Os/sec for random writes, as compared to 30 I/Os per second in the case of reads.

The slightly better performance is due to the ability of the disks to perform the seek simultaneously with the transfer of data into the disk buffer. This is not possible on random read operations.

The significant differences between Tables 5 and 6 illustrate the different ways reads and writes are performed on the Wren IV disks. For reads, the seek is performed after the disk has relinquished the SCSI bus; after the seek, the disk initiates a reconnect to transfer data. For writes, the seek is begun concurrently with the data transfer phase, and the disk holds onto the SCSI bus for much longer (16600 usec) than in the read case (1500 usec). Thus, for the read trace, the SCSI bus was only about 33% utilized, while in the write case is is 99% utilized.

The kernel traces shown as timelines in Tables 30 and 31 show that a single operation spends the same amount of time on the Jaguar board for random reads and writes. The difference in the operations, apparent in Tables 5 and 6, is how long an operation holds on to the SCSI bus. For small (4 KByte) random operations, this difference does not affect performance for write compared to reads, since the SCSI bus had low utilization in the read case. The writes "use up" the remaining bandwidth of the SCSI bus, and the resulting I/O rates in Figures 5 and 6 are very similar.

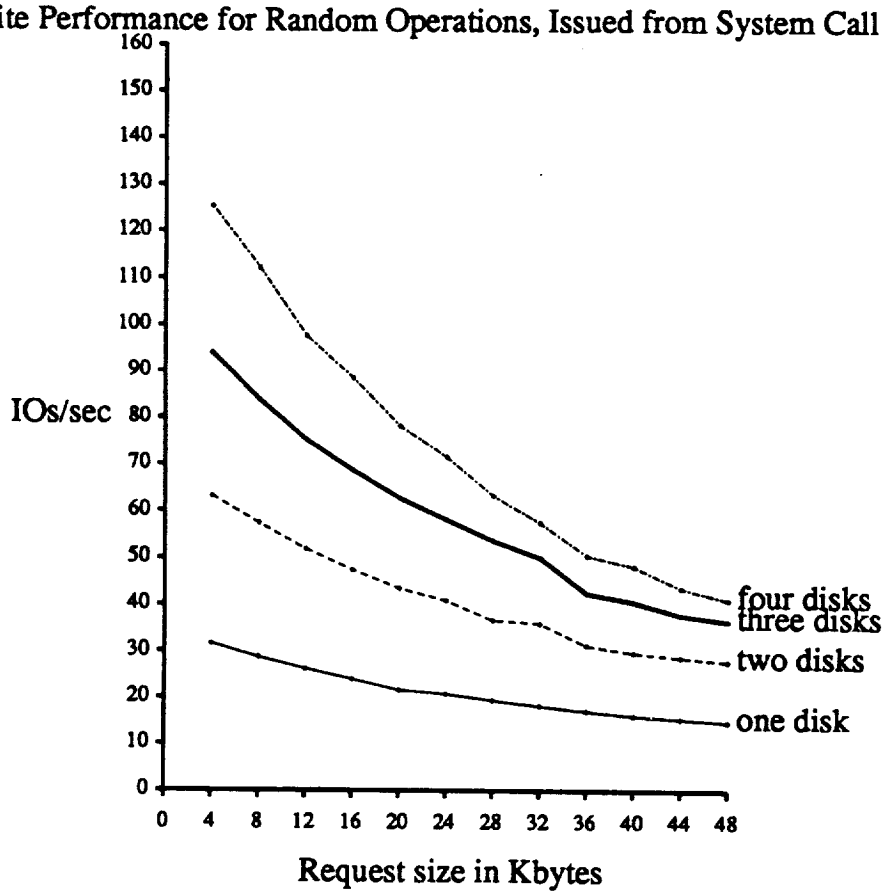


Figure 6: I/O Rates for Random Write Operations Issued from the No-Copy System call to four disks on a single string. A separate process issues requests to each active disk.

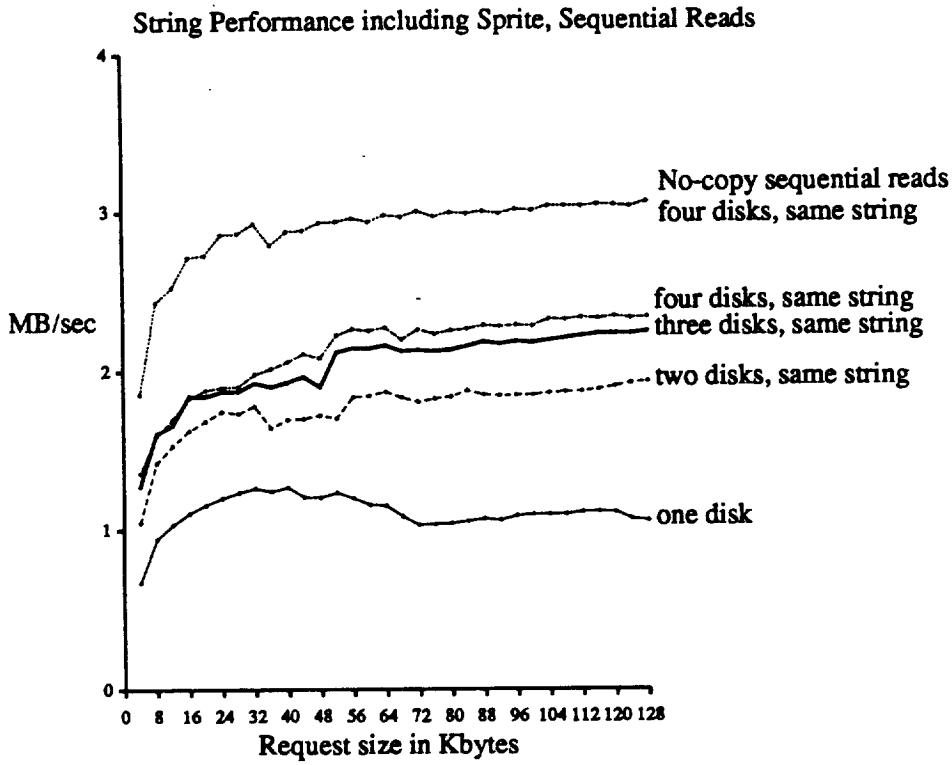


Figure 7: Bandwidth for sequential read operations issued from Sprite user level to four disks on a single string. A separate process issues requests to each active disk. The top line in the graph is the bandwidth for no-copy system call reads issued to four disks, included for comparison.

4.1.2 String Performance Including Sprite

This section looks at string performance when I/O operations are issued as raw disk requests from Sprite user level, rather than with the no-copy system call. The previous section represented the best performance that could be achieved on a string. There is an obvious drop in performance when executing a "real" operating system. Unlike the special no-copy system call, the Sprite operating system must return data to the user process that requested it. In order to do this, Sprite performs a number of copy and cache flush operations. These operations cause contention on the Sun4/280 host's memory system, resulting in performance that is lower than the maximum numbers described in the last section.

Sequential Reads

Figure 7 shows a graph similar to that of Figure 3. It depicts sequential read activity for one, two, three and four disks on a single SCSI string. Here, the requests are issued as raw disk requests from Sprite user level.

The performance of a single disk on the string is somewhat lower than the corresponding performance in Figure 3. The highest bandwidth achieved is 1.2 MBytes/sec for request sizes in

the range of 16 KBytes to 56 KBytes. For larger request sizes, the bandwidth drops to about 1 MByte/sec. There is no obvious explanation for the performance drop for larger requests. Tables 12 and 14 show the timelines for 128 KByte and 32 KByte sequential read operations. These timelines demonstrate that the time on the Jaguar board, the time for the DMA free cache flush operation and the data copy time are all proportional to the size of the transfer. Since as many bytes must be copied and flushed per MByte in either case, there appears to be no reason for the performance drop for larger sequential reads issued from Sprite user level. This decrease has been consistently measured over many iterations of the test, however.

The performance degradation for 32 KByte sequential reads issued under Sprite compared to those issued from the no-copy system call is the result of the extra processing necessary when Sprite is used. A comparison of the timelines in Tables 14 and 24 show that sequential reads issued from the no-copy system call and from Sprite spend the same amount of time on the Jaguar board. However, there is considerable extra processing evident in the Sprite timeline. In particular, significant time is spent doing the cache flush of DMA space and doing the copy of data from the kernel to user space.

When a second disk on the string is accessed along with the first, the combined bandwidth of the two disks is around 1.8 MBytes/sec, compared to approximately 2.6 MBytes/sec in figure 3. When a third and fourth disk are accessed, performance continues well under the level of Figure 3. The maximum bandwidth seen in the four disk cases is around 2.3 MBytes/sec, about 75% of the 3 MBytes/sec bandwidth achieved for sequential reads issued to four disks from the no-copy system call, shown in the top line of Figure 7 for comparison.

Table 7 shows the breakdown of time spent in the various SCSI phases for the 32 KByte raw sequential reads issued to four disks (one process issuing requests per disk). The SCSI bus is idle only 3.8% of the time in this trace. This number is slightly larger than the corresponding bus free time in Table 3, but indicates that the SCSI bus is highly utilized, and is the reason for the performance limitation of the string.

Tables 7 and 3 are surprisingly similar. The only significant difference is that a little more time is spent in the data transfer phase in the Sprite trace.

The overall throughput of this trace is limited to 2.3 MBytes/sec. This turns out to be an important number, since it is the maximum bandwidth that can be achieved by the array under Sprite on the Sun4/280. The host utilization, which measures 75.6% in this case, is the reason for this limitation. It indicates that the host memory bus is reaching saturation, and cannot support greater bandwidth. During Sprite I/O, the host performs copy operations and cache flushes. These operations saturate the host memory system, and cause the performance bottleneck at 2.3 MBytes/sec.

When the no-copy system call was used, the comparable host utilization for similar operations was 7.69%.

Sequential Writes

The performance of large sequential writes using full Sprite processing is shown in Figure 8, and the SCSI phases for a trace of four disks performing 32 KByte sequential writes are described in Table 8. Compared to Figure 4, where the operations were issued from the low overhead system call and four disks achieved 2 MBytes/sec, in Figure 8 only 1.5 MBytes/sec is achieved. This 25%

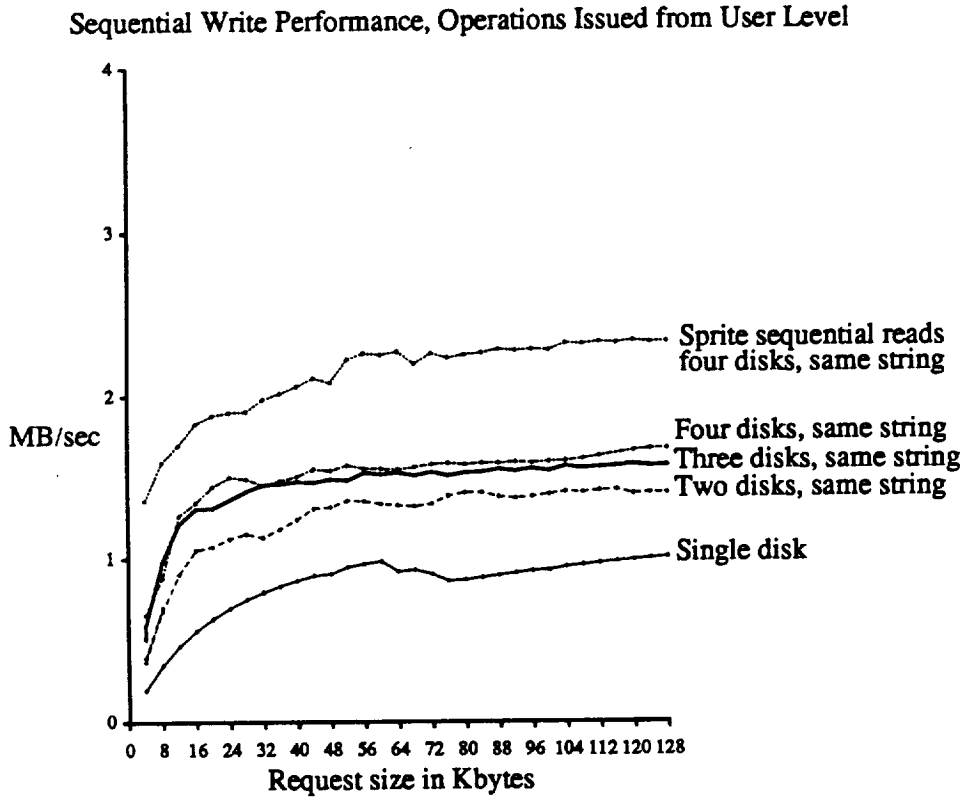


Figure 8: Bandwidth for sequential writes issued from Sprite user level. A separate process issues requests to each active disk. The top line in the graph is the bandwidth of sequential reads issued to four disks from Sprite user level, included for comparison.

difference in performance may be caused by a cache miss for the buffer from which data is being transferred on the host.

The performance of user level sequential writes is also significantly lower than user level sequential reads, shown in the top line in Figure 8. In the last section, we observed that the trace of 32 KByte sequential reads issued from the special system call had data transfer phases that averaged 13 msec, instead of the minimum time of about 8 msec. In the case of 32 KByte sequential writes issued from user level, the time spent in the data transfer phase averages 20 msec. A cache miss would explain this discrepancy.

As in the no-copy system call traces, the traces of sequential reads and writes issued from Sprite user level show that much more time is spent on the Jaguar board in a single user-level sequential write than in a read. Once again, this increase can be explained by a missed rotation suffered by sequential writes. In the user level I/O traces, the difference between the time spent on the board for reads and writes is 10 msec. This is less than the 16.7 msec required for a full rotation, since host processing overlaps about 6 msec of the disk rotation. This 6 msec of host processing can be seen in Table 14 as the sum of the times required to perform the data copy, the cache flush, and other Sprite operations.

The measured host utilization in this case is 36.3%, significantly higher than the 5.2% in the case where writes were issued from the special system call, but significantly lower than the 75.6

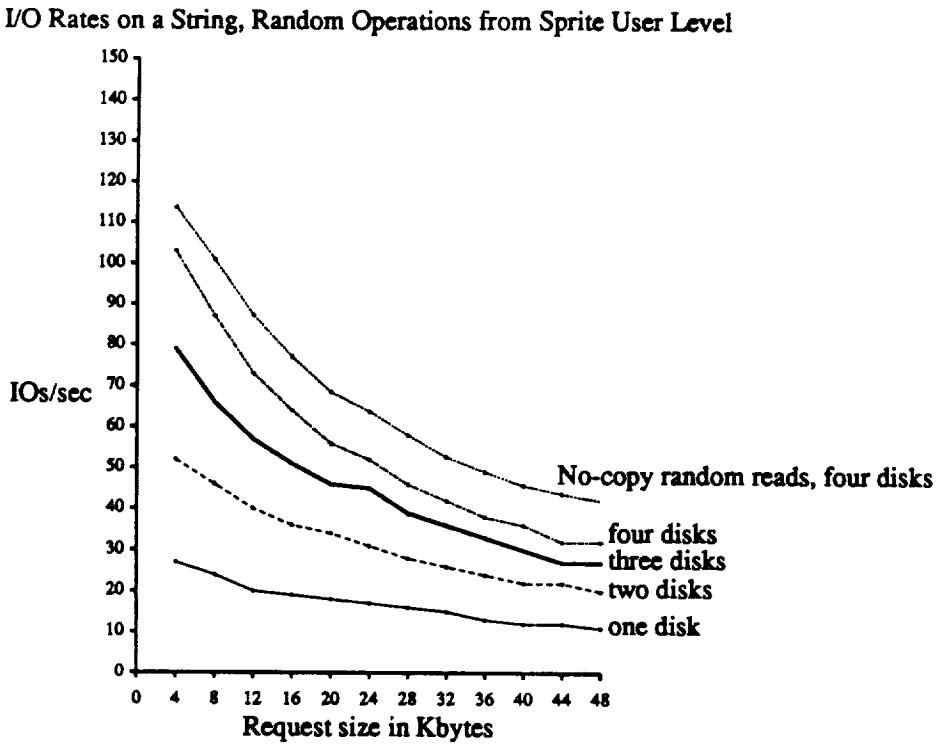


Figure 9: I/O Rates for small random reads issued from Sprite user level. A separate process issues requests to each active disk. The top line in the graph shows the I/O rates for small random reads issued from the no-copy system call, included for comparison.

% utilization for sequential reads. The reason that utilization is lower for writes than for reads is that the DMA flush operation for a sequential write operation is much faster than for a sequential read operation, since in the write case, the flushed cache blocks are likely to be clean. Tables 14 and 15 show that the DMA flush for reads is approximately five times slower in the read case. See the discussion in 4.7 for more details on cache flushing rates.

Random Reads

Figure 9 shows the I/O rates achieved for small random reads issued from Sprite user level. Compared to Figure 5, this graph shows a small performance penalty for small random operations issued from Sprite user level. This penalty translates into a loss of 2 or 3 I/Os per second per active disk when the random operations are issued from Sprite user level, or about 10% per disk. In the case of four disks performing 4 KByte random reads on a string, issuing the reads from Sprite user level results in an I/O rate of 103 I/Os/sec, while issuing them from the no-copy system call gave 114 I/Os/sec. The top line in Figure 9 shows no-copy read I/O rates for comparison.

This small performance degradation is due to the addition of full Sprite processing, including copy and cache flush operations. The copy overhead is proportional to the size of the transfer, and is relatively small for 4 KByte operations, so subsequent operations are issued fairly quickly. There

is some delay in issuing subsequent operations, however, so the disks are not kept as busy as they were when I/O was issued from the special system call.

Table 9 shows the breakdown for small random reads issued from Sprite user level. In comparison to Table 5, Table 9 shows a slightly higher percentage of bus free time. This is evidence that there is some effect from the delay in issuing subsequent instructions due to the data copy operation. The number of I/Os per second is also reduced a little for the case of a single process issuing requests to a particular disk; this number drops to 121 from 130 in the earlier case. However, having multiple processes issue requests to the four disks (to increase the depth of the request queues) brings the total number of I/Os/sec back to previous levels, and to the maximum capabilities of the disks.

Wren IV	
Diameter	5 1/4"
Formatted Capacity	344 MBytes
Rotational Speed	3600 RPM
Avg. rotation	8.33msec
Avg. seek	17.5msec
Surfaces	9
Heads	9
Tracks	549 per surface
Sectors per track	variable
Data transfer rate	variable, avg. 1.3 MB/sec
I/O rate (4KByte random reads)	33 I/Os/sec
Bytes per sector	512 default

Table 1: Characteristics of Imprimis Wren IV Disks Used in RAID-I

Sprite Instruction Counts	
Trap for read system call	118 instr
Read system call; convert file handle to pointer	536 instr
Look for request in cache	192 instr
On cache miss, convert data request into disk block address; generate "generic" SCSI command	1228 instr
Setup command in Jaguar device driver; convert to Jaguar-specific command format; stuff into Jaguar board; allocate DMA space for data to be written into (I/O issued at this point)	951 instr
Back out of routines to wait for interrupt from disk	443 instr
On interrupt, check the operation's status; free DMA space used for data	3548 instr
Copy 4K to user space, back out of procedure calls (includes 20 register window overflow/ underflow pairs)	6433 instr
Total: 13449 instr	

Table 2: Instruction counts for issuing a 4 KByte file system read operation from Sprite user level.

Time in Each SCSI Phase	Percentage	Normalized
Arbitration	.06 %	6.5 usec
Selection	.20 %	22 usec
Command	4.02 %	430 usec
Message Out	.72 %	77 usec
Message In	.22 %	24 usec
Data Transfer	93.15 %	10000 usec
Disconnect/Reconnect	0.11 %	12 usec
Reselection	0.02 %	2.0 usec
Status	0.45 %	48 usec
Bus Free	1.05 %	
Elapsed Time of Trace	6.64 sec	
I/Os completed	620	
Bandwidth	2.92 MBytes/sec	

Table 3: Breakdown of time in SCSI phases for trace of 32 KByte sequential reads issued by no-copy system call. Four disks are active during this trace, with a separate process issuing requests to each disk. Note the high (99%) utilization of the SCSI string. The column labeled "Normalized" gives the average time per I/O spent in each phase of the protocol.

Time in Each SCSI Phase	Percentage	Normalized
Arbitration	0.07 %	11 usec
Selection	0.14 %	22 usec
Command	2.3 %	355 usec
Message Out	0.16 %	25 usec
Message In	1.9 %	290 usec
Data Transfer	93.1 %	14400 usec
Disconnect/Reconnect	1.06 %	160 usec
Reselection	0.21 %	32 usec
Status	0.31 %	48 usec
Bus Free	0.77 %	
Elapsed Time of Trace	8.73 sec	
I/Os completed	565	
Bandwidth	2.02 MBytes/sec	

Table 4: Breakdown of time in SCSI phases for trace of 32 KByte sequential writes issued by special no-copy system call. Four disks are active in this trace, with a separate process issuing requests to each disk.

Time in Each SCSI Phase	1 Process/Disk		4 Processes/Disk	
	Percentage	Normalized	Percentage	Normalized
Arbitration	0.43 %	33 usec	0.37 %	28 usec
Selection	0.55 %	42 usec	0.52 %	39 usec
Command	6.21 %	480 usec	6.33 %	480 usec
Message Out	4.29 %	330 usec	4.37 %	330 usec
Message In	0.32 %	25 usec	0.32 %	24 usec
Data Transfer	20.0 %	1500 usec	20.4 %	1500 usec
Disconnect/Reconnect	0.38 %	29 usec	0.38 %	29 usec
Reselection	0.07 %	5.4 usec	0.06 %	4.7 usec
Status	0.68 %	52 usec	0.68 %	52 usec
Bus Free	67.1 %		66.6 %	
I/O's per second	130		132	

Table 5: Breakdown of time in SCSI phases for two traces of 4KByte random reads issued from the no-copy system call. Four disks are active in each case. The first two columns show statistics for requests generated by a single process per disk, and the second two show statistics for requests generated by four processes per disk.

Time in Each SCSI Phase	Percentage	Normalized
Arbitration	0.2 %	35 usec
Selection	0.2 %	30 usec
Command	2 %	360 usec
Message Out	0.1 %	22 usec
Message In	1.8 %	330 usec
Data Transfer	94.0 %	16600 usec
Disconnect/Reconnect	0.6 %	100 usec
Reselection	0.005 %	0.8 usec
Status	0.3 %	48 usec
Bus Free	0.7 %	
I/O's per second	*	

Table 6: Breakdown of time in SCSI phases for trace of 4KByte random writes issued from No-Copy system call. Four disks are active in this trace, with a separate process issuing request to each disk. *Because of recent problems with the trace upload program, these trace statistics had to be gathered by hand over a small number of I/Os. They are less reliable than the other tables, so I don't include an overall I/O rate number here.

Time in Each SCSI Phase	Percentage	Normalized
Arbitration	0.05 %	7.0 usec
Selection	0.16 %	22 usec
Command	3.0 %	420 usec
Message Out	0.49 %	68 usec
Message In	0.17 %	24 usec
Data Transfer	91.9 %	12800 usec
Disconnect/Reconnect	0.07 %	9.8 usec
Reselection	0.01 %	1.4 usec
Status	0.41 %	57 usec
Bus Free	3.8 %	
Elapsed Time of Trace	8.72 sec	
I/Os complete	625	
Bandwidth	2.24 MBytes/sec	

Table 7: Breakdown of time in SCSI phases for a trace of 32 KByte sequential reads issued from Sprite user level. Four disks are active in this trace, with a separate process issuing requests to each disk.

Time in Each SCSI Phase	Percentage	Normalized
Arbitration	0.05 %	10 usec
Selection	0.10 %	22 usec
Command	1.68 %	370 usec
Message Out	0.11 %	24 usec
Message In	1.25 %	280 usec
Data Transfer	89.3 %	19700 usec
Disconnect/Reconnect	0.77 %	170 usec
Reselection	0.14 %	30 usec
Status	0.22 %	49 usec
Bus Free	6.41 %	
Elapsed Time of Trace	12.5 sec	
I/Os completed	566	
Bandwidth	1.42 MBytes/sec	

Table 8: Breakdown of time in SCSI phases for trace of 32 KByte sequential writes issued from Sprite user level. Four disks are active in this trace, with a separate process issuing requests to each disk.

Time in Each SCSI Phase	Percentage	Normalized	Percentage	Normalized
Arbitration	0.12 %	9.6 usec	0.42 %	33 usec
Selection	0.27 %	22 usec	0.52 %	40 usec
Command	5.66 %	470 usec	6.16 %	480 usec
Message Out	3.60 %	300 usec	4.38 %	340 usec
Message In	0.30 %	24 usec	0.31 %	24 usec
Data Transfer	18.7 %	1500 usec	19.7 %	1500 usec
Disconnect/Reconnect	0.53 %	44 usec	0.38 %	30 usec
Reselection	0.04 %	3.2 usec	0.06 %	4.7 usec
Status	0.62 %	51 usec	0.68 %	53 usec
Bus Free	70.2 %		67.4 %	
I/O's per second	121		129	

Table 9: Breakdown of time in SCSI phases for two traces of 4KByte random reads issued from Sprite user level. The first two columns show statistics for requests generated by a separate process, and the second for requests generated by four processes per disk.

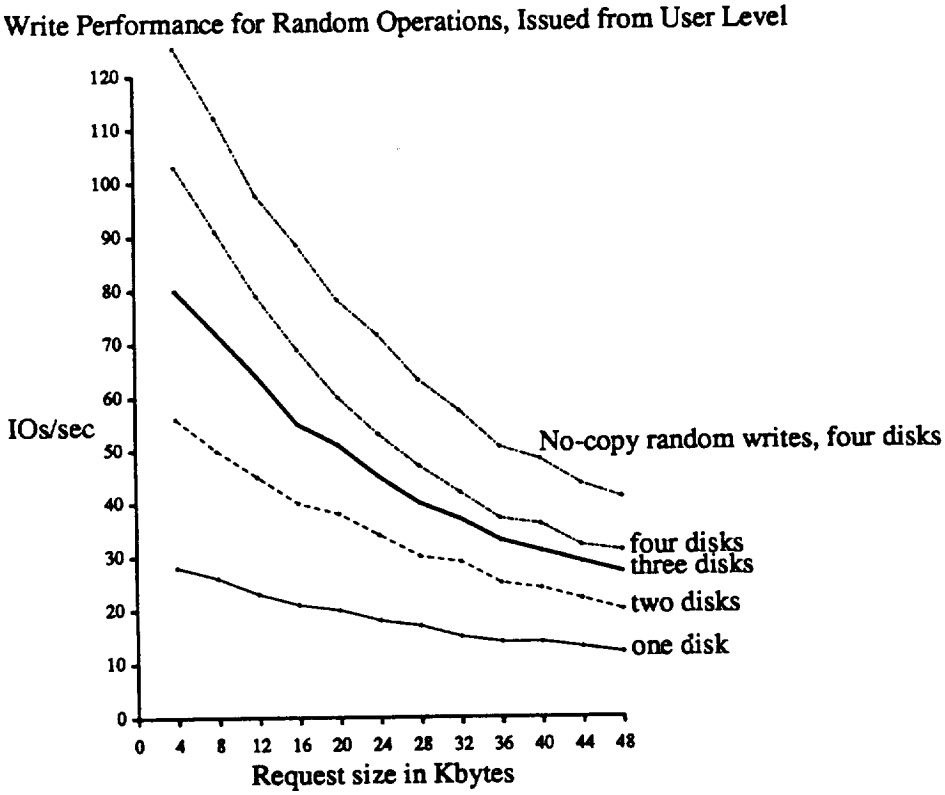


Figure 10: I/O Rates for small random writes issued from Sprite user level. A separate process issues requests to each active disk. The top line in the graph shows the I/O rates for small random writes issued from the no-copy system call, included for comparison.

The Sun4/280 host utilization numbers for small random read operations issued to four disks from Sprite user level is 21.3%. Compared to the 7.0% utilization when requests were issued from the no-copy system call, the utilization increase caused by including all Sprite processing is significant.

Random Writes

Figure 10 shows I/O rate performance for small random write operations issued from Sprite user level. The penalty for including Sprite is about the same as in the random read case: close to 10% of the I/O rate per disk for small operations. The no-copy small random write performance line is included in the graph for comparison. Table 10 shows the breakdown of time spent in the various SCSI phases. Performance for random writes issued from Sprite is very close to that for random reads.

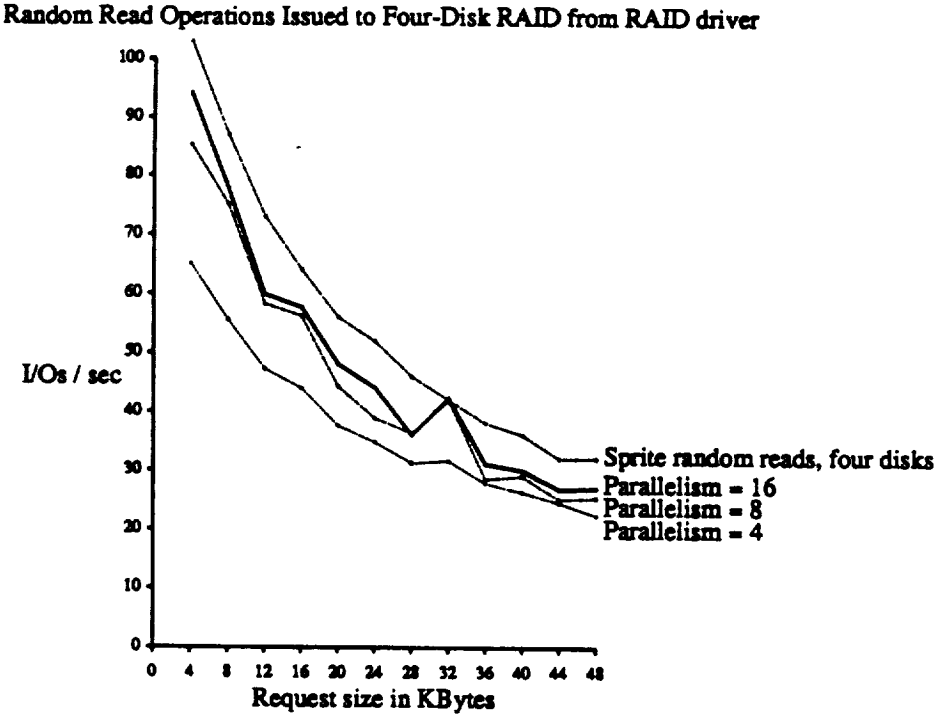


Figure 11: I/O Rates of four disks on one string configured as a single-row RAID device performing random read operations. Parallelism of 4, 8 and 16 indicates the number of independent processes issuing requests to the array. The top line in the graph shows small random reads issued from Sprite user level; it is included for comparison.

4.1.3 String Performance including Sprite and the RAID Driver

Figures 11 and 12 show the performance of random read and write operations performed on four disks on a string configured as a four-disk, single row RAID Level 5 logical device [RAID]. Requests are issued to these disks using the RAID driver written by Ed Lee. Data is striped across the disks in units of 32 KBytes. Requests of less than 32 KBytes are satisfied by a single disk. (Requests are aligned on stripe unit boundaries.)

It is difficult to compare the performance of disks accessed directly with those accessed as part of a RAID device. The goal in including these numbers is to demonstrate that the extra overhead of executing the RAID driver has a significant effect on I/O rates, as seen in the top lines of Figures 11 and 12.

I tested three different workloads: parallelism of 4, 8 and 16 processes issuing I/O requests to the 4-disk RAID at a time. Since the sectors requested were chosen randomly, more processes issuing requests at a time increased the likelihood of disks being highly and evenly utilized. The RAID device achieves about 95 I/Os per second in the random read case with parallelism of 16 and request size of 4 KBytes. This compares to approximately 105 I/Os per second achieved when requests were issued from Sprite user level.

Write performance is poor in this case, since we are performing small writes. On a RAID device, a small write turns into four separate operations, since the old data must be read along

Time in Each SCSI Phase	Percentage	Normalized
Arbitration	0.12 %	26 usec
Selection	0.25 %	55 usec
Command	4.9 %	1080 usec
Message Out	0.27 %	60 usec
Message In	2.6 %	570 usec
Data Transfer	16.9 %	3700 usec
Disconnect/Reconnect	1.88 %	410 usec
Reselection	0.09 %	20 usec
Status	0.53 %	120 usec
Bus Free	72.4 %	
I/O's per second	111	

Table 10: Breakdown of time in SCSI phases for trace of 4KByte random writes issued from Sprite user level. A separate process issues requests to each active disk.

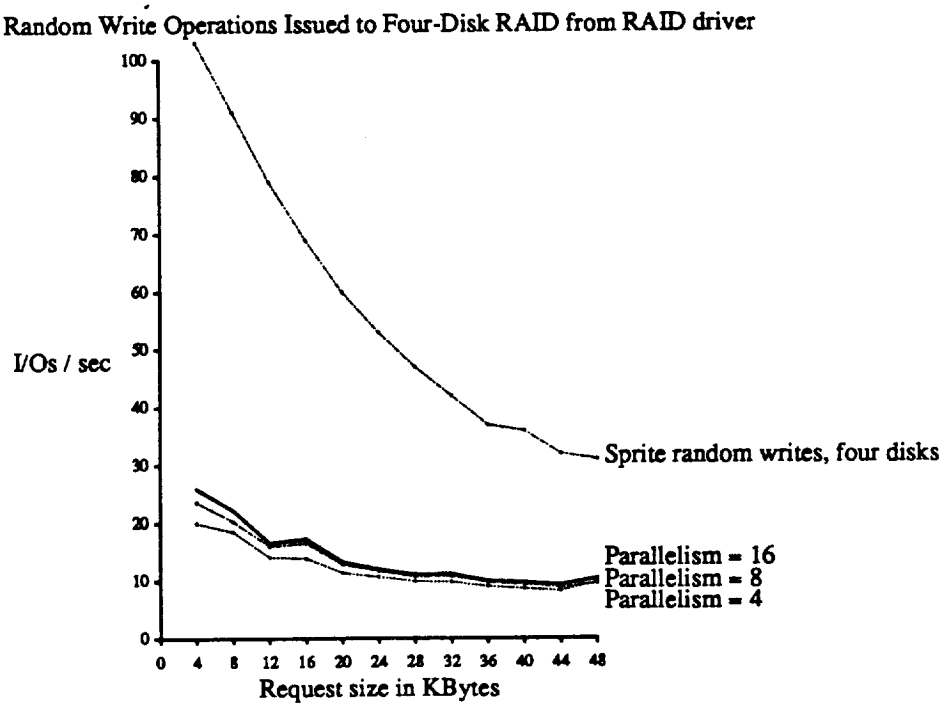


Figure 12: I/O Rates of four disks on one string configured as a single-row RAID device performing random write operations. Parallelism of 4, 8 and 16 indicates the number of independent processes issuing requests to the array. The top line in the graph shows small random writes issued from Sprite user level; it is included for comparison. Note that the I/O rates for the RAID device are logical rather than physical I/Os, and actually represent four physical I/Os for each logical I/O.

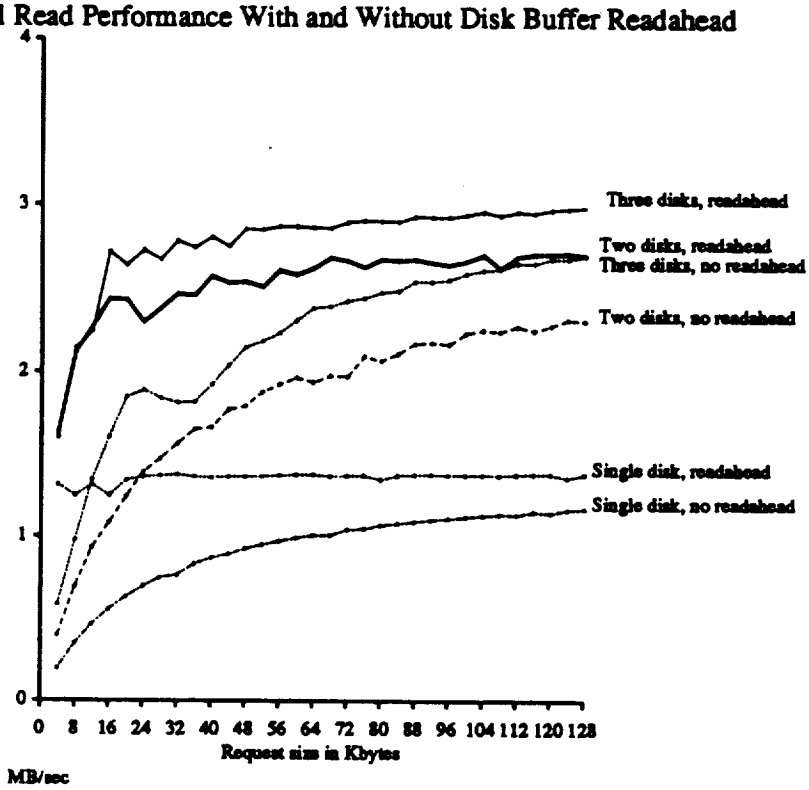


Figure 13: Bandwidth with and without readahead enabled for sequential read operations generated using the no-copy system call to three disks on a single string. A separate process issues requests to each active disk.

with the old parity, and when the parity has been updated, the new data and new parity must be written. See [RAID] for a discussion of the small write performance issues in RAID. The small write performance was about 27 (logical) small writes/sec, compared to 111 (physical) small writes/sec performed from Sprite user level.

4.2 Disk Parameters

Very few parameters on the Wren IV disks can be varied by the user. Those that can include the readahead enable bit, the Buffer Full Ratio and Buffer Empty Ratio. The following sections describe the effect on performance of varying these parameters.

4.2.1 Using the Disk Buffer for Readahead

The Imprimis Wren IV disks have 32 KByte buffers through which all data is transferred. These buffers are used for speed matching, allowing the disk to transfer data across the SCSI bus at 4 MBytes/sec instead of at the rate data comes off the disk head (1.3 MBytes/sec). In addition, these buffers can be enabled to perform readahead to improve the performance of sequential read operations.

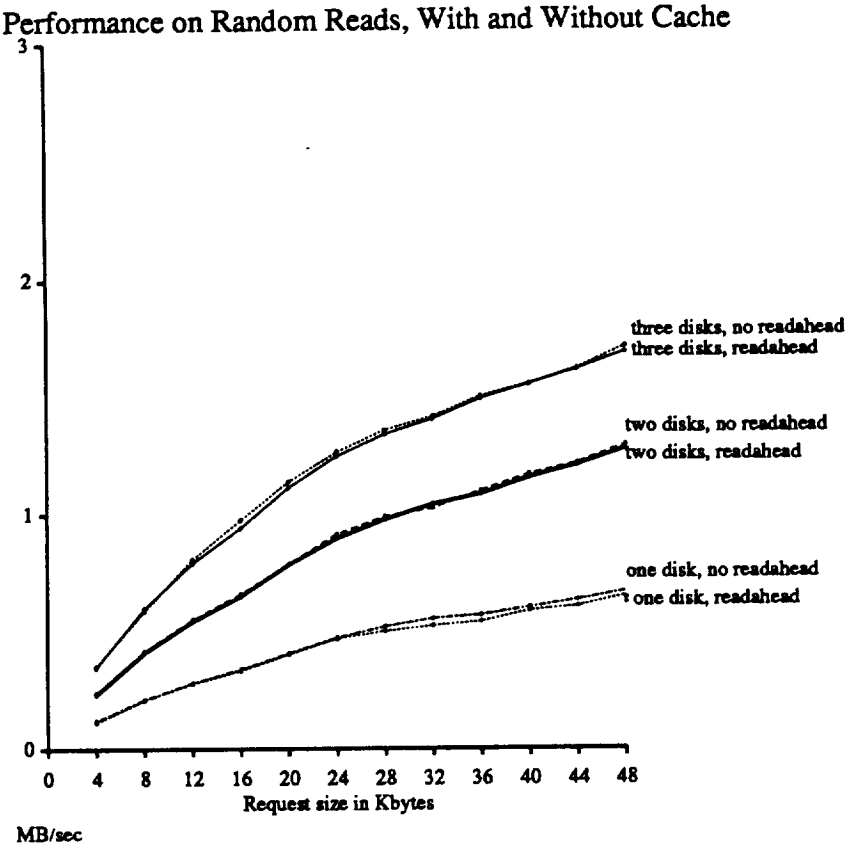


Figure 14: I/O Rates with and without readahead enabled for random reads generated using the no-copy system call to three disks on a string. A separate process issues requests to each active disk.

Figure 13 shows the performance of various numbers of disks with and without their disk buffers enabled for readahead on sequential read operations. This test used the no-copy system call. The gap in performance between enabling the disk buffers for readahead and not doing so is consistently large for sequential operations. It is most dramatic on small requests, since when readahead is enabled, these can come directly from the track buffer without waiting for additional disk accesses. In contrast, as requests get larger than the full track buffer size (32K), the performance advantage for readahead on the disk buffer narrows. On these larger operations, the track buffer will have to be filled multiple times, and readahead will give an advantage only for the first 32 KBytes of a request. This explains why the gaps in performance narrow as requests get large.

While track buffers enabled for readahead improve the performance of sequential reads, they degrade performance very slightly for random reads. Figure 14 shows small random read operations on a string with between one and four disks active, with and without readahead. The performance difference is very small. Where there is a difference, the advantage generally goes to the case where the track buffer is not enabled for readahead. Not surprisingly, it appears to take a little longer to stop writing into the track buffer and perform a seek than it does to perform the seek when readahead is not being performed. The performance difference is shown in the graph.

The advantage for sequential read operations is so large and the performance penalty for random operations is so small that there is no reason to disable the readahead option on the Wren IV disk buffers.

4.2.2 Buffer Full and Empty Ratio Effects

The role played by the Buffer Full Ratio (BFR) and the Buffer Empty Ratio (BER) was described in Section 2.1.3. The BFR relates to reads, and specifies the amount of data that must be read into the disk buffer before the disk attempts to obtain the SCSI bus to transfer data. The BER relates to writes; after the data buffer fills with data to be written to disk, the disk will disconnect from the SCSI bus and write the data from the buffer onto the disk medium. When the disk's buffer is empty enough to meet the BER, then a reconnection will occur and more data will be accepted by the disk from the HBA. (For simplicity, this section will refer to BFR and BER values by the number of bytes represented by the ratio, rather than by the ratio itself.)

The choice of the BFR and BER parameters determines the number of disconnects and reconnects that will occur during an I/O. A small BFR (say, 512 bytes) will result in a large number of disconnects during a read operation. Whenever the disk reads 512 bytes into its buffer, the disk will attempt to get hold of the bus and transfer the data out. During the time it takes to arbitrate for the bus, some more data for the request will be read into the buffer. Data can only be written into the buffer at a rate of 1.3 MBytes/sec, but it is transferred across the SCSI bus at 4 MBytes/sec. Thus, the buffer will soon be emptied, and a disconnect will occur. The smaller the BFR, the more frequent are the disconnects and the shorter the time a particular device will hold onto the SCSI string to transfer data, since there will be less data in the buffer to transfer out. Analogous behavior occurs for writes depending on the value of the BER.

From the SCSI phase tables of Sections 4.1.1 and 4.1.2, we saw that disconnect, reconnect, arbitration and messages for saving and restoring data pointers accounted for a very small part of the lifetime of a transaction. Since disconnects are not "expensive", we would expect that changing the BFR and BER would not have much effect. This turns out to be true in the middle range of possible values for the BFR and BER. However, there are some minor differences that result from changes in the BFR and BER in this range. Below, I explain the expected performance differences for different BFR values and show experimental data to support these predictions. (The remainder of this discussion will focus on reads and the BFR. Analogous conclusions apply for writes and the BER.)

Small random (e.g., 4 KByte) operations are dominated by their seek times. Good performance on such operations depends on how quickly their seeks are issued. A large BFR allows one transaction to transfer all its 4 KBytes of data before giving up the SCSI bus. A small (e.g., 512 bytes) BFR, on the other hand, results in a number of disconnects during the 4 KByte data transfer, allowing seek operations for other disks to be issued more quickly. Thus, somewhat better performance for small random operations is expected for smaller BFRs.

By contrast, good performance for large operations depends on efficient data transfer. Large BFRs are expected to do better for such workloads, since they allow a large amount of data to be transferred for each SCSI disconnect/reconnect overhead. A small BFR for a very large (e.g., 128 KByte) transfer would result in a large number of disconnects and reconnects. Although the overhead for each disconnect and reconnect is small, the combination of so many is expected to be significant enough to affect performance.

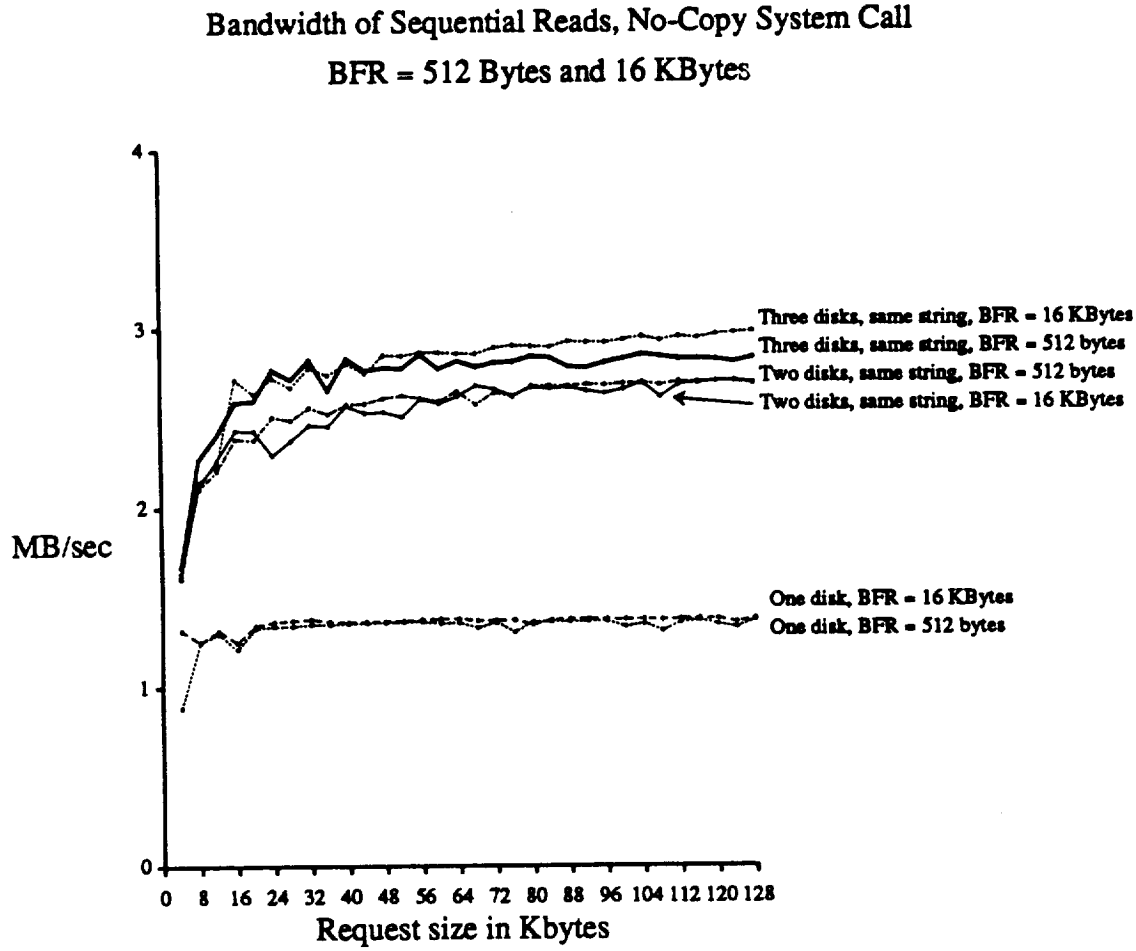


Figure 15: Bandwidth for sequential reads generated using no-copy system call for BFR = 16 KBytes and BFR = 512 bytes.

These expectations generally agree with observed performance. Figure 15 shows sequential read performance for one, two and three disks on a single string for BFR = 512 bytes and for BFR = 16 KBytes. These I/Os were generated using the no-copy system call. For one and two disks active on a string, there is not a clear winner between the BFR values, although the smaller BFR is ahead for more of the time. In the three-disk case, the smaller BFR still wins on smaller operations (up to about 32 KBytes) because the latency is lower with the smaller BFR. However, for larger operations, the 16KByte BFR is a clear winner. As expected, in these larger operations the extra disconnects and reconnects start to hurt performance for when the BFR is small, while in the large BFR case, transfers are efficient.

For random reads, Figure 16 shows that the difference in performance between the two BFRs is consistently very small. However, in the small random case, the advantage consistently goes to the smaller BFR. The reason for this is the same as mentioned above: for smaller operations, some performance is lost if the BFR is too high, and one disk keeps the SCSI bus during data transfer long enough that the latency of other operations is affected.

It is clear from the graphs that varying the BFR over the range of values shown doesn't have much effect on performance. A BFR (or BER) in the middle range of possible values will make little difference to overall throughput.

One BFR does result in dismal performance. Figure 17 shows the performance for one, two and three disks performing sequential reads with a BFR = 32 KBytes. This corresponds to the entire disk buffer on the Wren IV. When the entire buffer must be full before data transfer is initiated, there are two effects. The first is that while arbitration, reselection and identification are going on (a process that takes hundreds of microseconds), no more data can be transferred into the disk buffer from the disk. For any BFR except the largest, the overhead of reselection is overlapped with data continuing to fill the disk buffer. By the time reselection is complete, more data than the amount specified by the BFR is actually in the disk buffer, ready to be transferred to the host. No such overlap of reselection overhead and writing into the disk buffer is possible when the BFR requires filling the entire buffer before initiating the reselection. The second effect of the BFR = 32 KBytes is that very large latencies are introduced for any operations waiting to complete while data is being transferred from the disk.

4.3 HBA Performance

Section 4.1.1 described the performance limitations of a single SCSI string. Another performance limitation in RAID-I is the amount of bandwidth that can be sustained by a single Jaguar Host Bus Adaptor.

Figure 18 shows evidence for this HBA bottleneck. The string limitation of Section 4.1.1 is apparent in the figure when three disks on a single string do not achieve much of a performance improvement over two disks. Moving one of the three disks to the second string controlled by a single HBA results in better performance than the single string case, since much of the string contention is alleviated. However, the performance achieved is less than the expected value of 3.9 MBytes/sec (three times the bandwidth possible on a single disk).

The top line on the graph indicates that the performance limitation is due to the capacity of the HBA. If three disks are placed on three separate Jaguars, the full performance possible on each disk is achieved. Again, the HBA has been regarded as a "black box" in this study; this bottleneck is observed rather than explained.

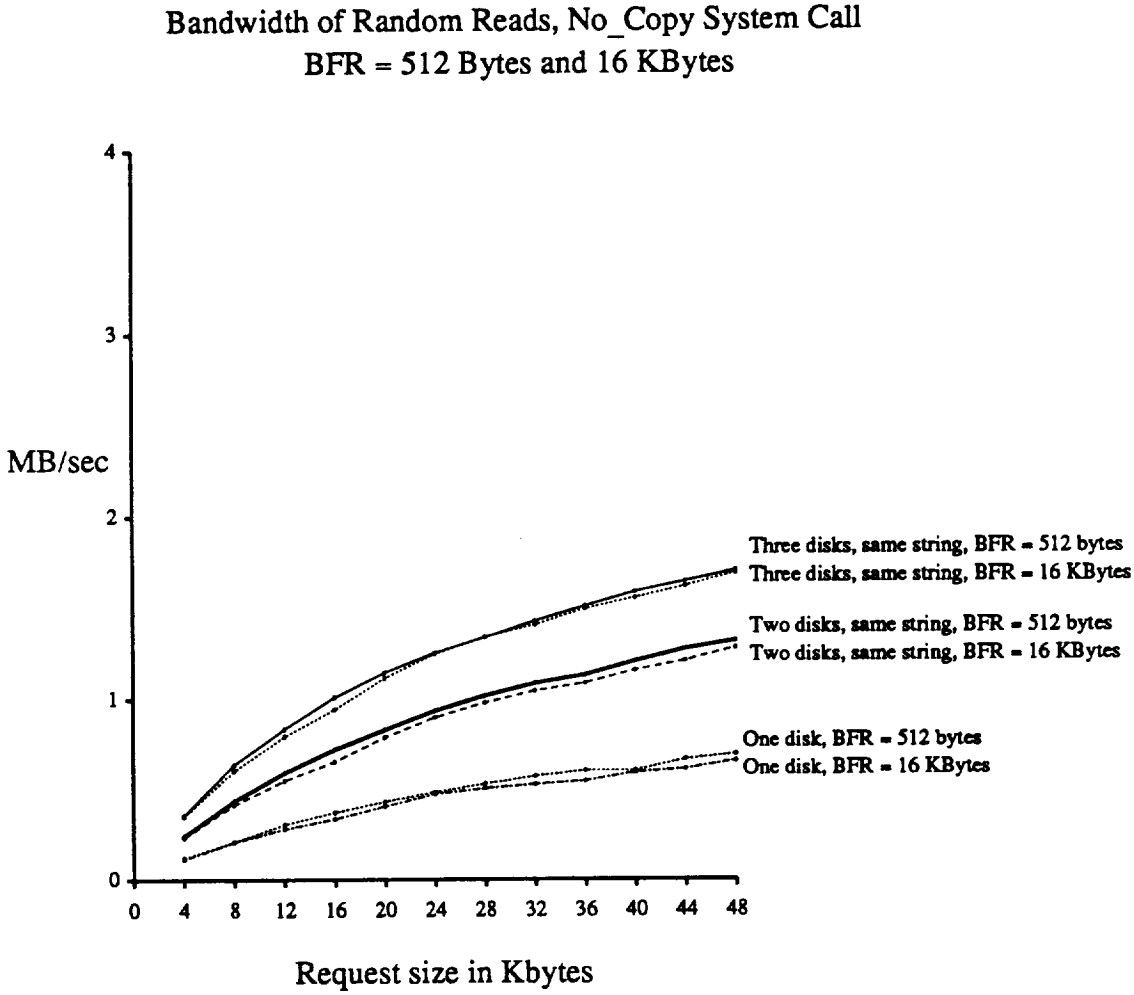


Figure 16: Bandwidth for random reads generated from the no-copy system call for BFR = 512 bytes and BFR = 16 KBytes.

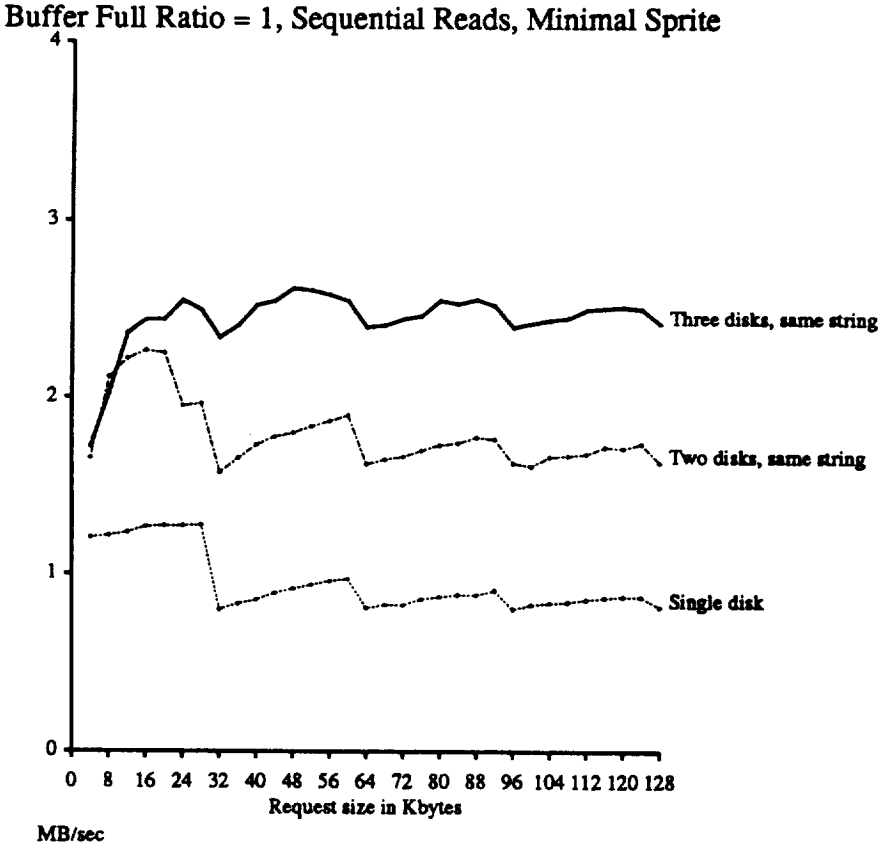


Figure 17: Bandwidth for sequential reads issued from no-copy system call, BFR = 32 KBytes

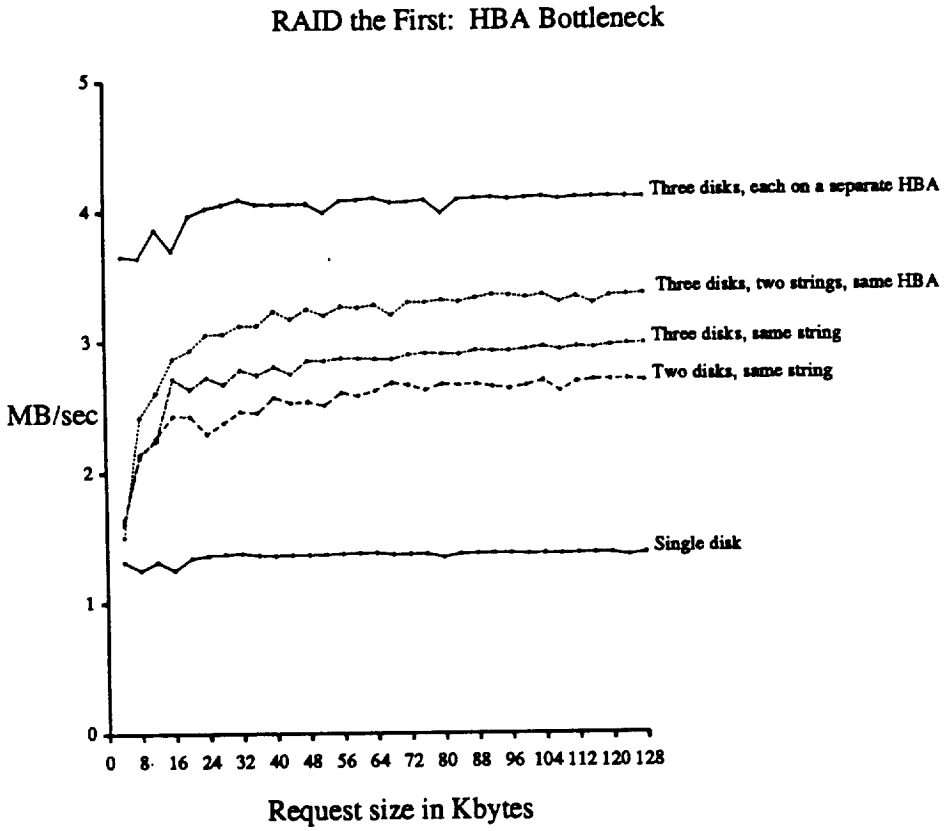


Figure 18: Bandwidth for sequential reads generated with the no-copy system call for one, two and three disks arranged in various ways on the two strings of a single HBA, and three disks on three HBAs. In each case, a separate process issued requests to each active disk.

Interphase confirmed that a single Jaguar HBA is capable of processing a maximum of 4 MBytes/sec total from the two strings attached to it. Since **each** SCSI string is nominally capable of achieving 4 MBytes/sec (or 5 with some other SCSI disks), it is clear that the Jaguar was not really designed to support high bandwidth. Rather, it was designed for more traditional systems requiring a high I/O rate. The differences in our design goals and those of the designers of some of the components used in RAID-I are discussed in Section 6.

4.4 Overall Sequential RAID Performance

Figure 19 shows the performance of the disk array for sequential reads of size 32 KBytes, when up to 13 disks are active in the array at a time. Programs that generated I/O activity on particular disks were activated in a round-robin fashion on the strings, to avoid as much string contention as possible. The tests for user level I/O used 11 disks on three strings, while the tests for the no-copy system call I/O used 13 disks on four strings.

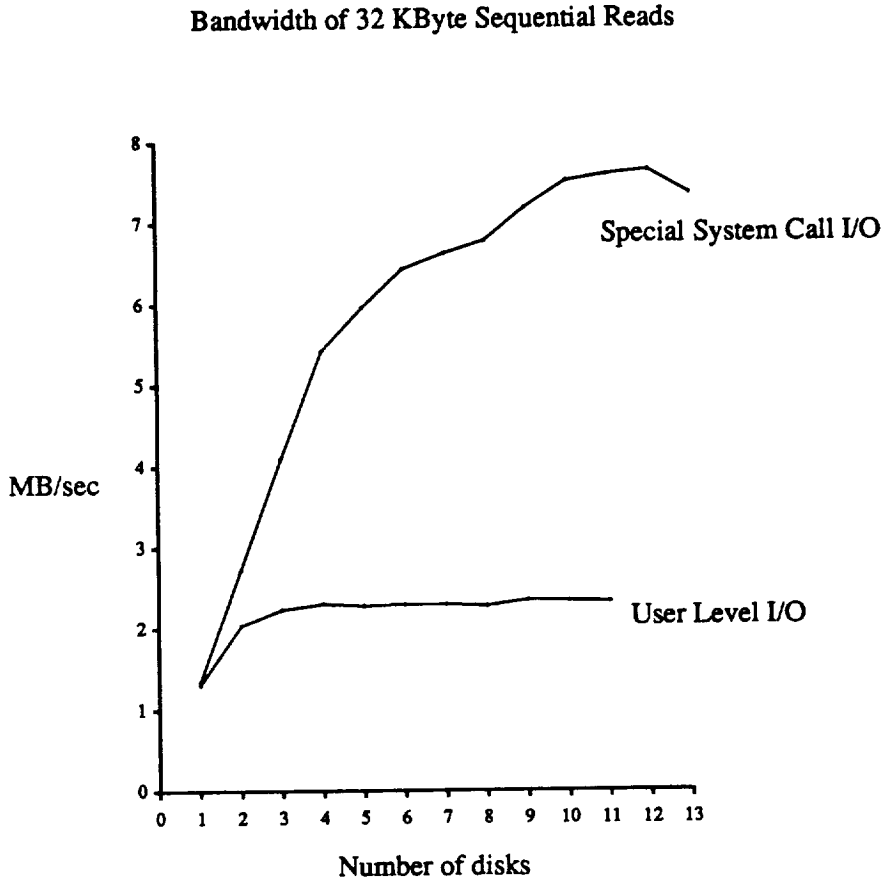
The lower line in the graph represents I/O operations issued from Sprite user level. It is obvious that the bandwidth possible when sequential read requests are issued from Sprite user level is limited to 2.3 MBytes/sec, no matter how many disks are active. Since in the case when requests are issued from user level, most of the time is spent performing copy operations and cache flushes, we believe that the Sun4/280 host's memory system is the cause of this performance limitation, rather than the Sun4/280 CPU.

As explained earlier, host utilization numbers provide a way of comparing the relative utilization of the Sun4/280's CPU and memory systems across the different workloads studied. The measured utilization when 11 disks are actively performing 32 KByte I/Os issued from user level is 97.3%, indicating that the memory system is completely saturated, and explaining the overall limitation of 2.3 MBytes/sec.

The other line in the graph shows I/O operations generated by the special no-copy system call for minimal Sprite processing. The graph rises steeply at first, with each disk providing its maximum bandwidth. After 4 disks are active in the array, performance increases begin to level off. For 13 active disks, approximately 7.5 MBytes/sec is delivered by the array. This bandwidth is lower than anticipated. This performance limit is not caused by the Sun4/280 host utilization, which measures 23.78% when 13 disks are active. This value is larger than any of the numbers seen in Section 4.1.1, but is still relatively low. The performance limit is also not caused by the string or HBA bottlenecks, since each of the string/HBA pairs should be able to deliver 3 MBytes/sec for a total of 12 MBytes/sec from the system.

Instead, the bottleneck is the result of saturation of the VME backplane. This is a surprise, since we expected closer to 10 MBytes/sec before the VME became a bottleneck in RAID-I. However, observing the behavior of the system when 13 disks are active shows that the VME bottleneck has arisen earlier. By the time 8 disks are active on four strings, the VME/HBA connection with the lowest priority on the backplane experiences so much delay in getting access to the VME that the HBA begins to experience timeouts. As further evidence that the VME is a bottleneck, the disk operations can be observed to complete in the exact order of their priorities on the SCSI bus and on the VME backplane, which was not the case earlier when the VME bus was not particularly heavily utilized.

The highest bandwidth ever measured on RAID-I was 9 MBytes/sec for 16 active disks performing 128 KByte sequential reads.



Special System Call I/O: 13 disks, activated round-robin, on 4 strings, 4 HBAs

User Level I/O: 11 disks, activated round-robin, on 3 strings, 3 HBAs

Figure 19: Bandwidth for 32 KByte Sequential Read operations for up to 13 disks over four strings (each string on a separate HBA) in RAID-I. The top line shows performance for I/Os generated from the no-copy system call, and the bottom line for those issued from Sprite user level.

4.5 I/O Rates

Figure 20 shows I/O rates achieved on 14 disks on four strings, each on a different HBA, performing small (4 KByte) random reads, where the I/O operations were issued from Sprite user level and from the no-copy system call.

The graph for operations issued from the no-copy system call is close to linear, increasing approximately 30 I/Os per second per disk to 420 I/Os per second for 14 disks. The linear increase shows that there is no bottleneck in this system other than what the disks themselves are capable of delivering. The measured host CPU/memory bus utilization in the 14-disk case is 40%, but it doesn't affect the number of I/Os per second achieved in the system.

However, when the requests are issued from Sprite user level, the I/O rates delivered by 14 disks are significantly lower. The first disk on each string contributes about 25 I/Os per second. The number of I/Os per second per disk decreases as more disks are added. 14 disks achieve approximately 275 I/Os per second, close to 20 I/Os per second per disk.

The host CPU/memory bus utilization measured for 14 active disks is 78.4%. This high utilization is not solely the result of memory system contention due to copy and DMA cache flush operations. We saw in the last section that Sprite could sustain 2.3 MBytes/sec of such activity, and the 14 disks performing 275 I/Os per second generate only 1.1 MBytes/sec of bandwidth. High host utilization in this case is mainly caused by the host CPU, which is required to perform 275 context switches per second. (A context switch takes about 1 msec in Sprite.)

Although we are encountering this host utilization limitation, we consider the performance achieved by RAID-I on small random operations (275 I/Os per second) to be excellent. RAID-I and Sprite deliver good performance for the small operations typical of current operating systems and databases.

4.6 Measured SCSI Overheads

The Wren IV disk and the Interphase Jaguar HBA each contain a processor and a SCSI chip that together implement the SCSI protocol. This implementation includes buffer allocation, saving and restoring state, and controlling the REQ/ACK lines for data transfer. This section observes the time required by each component for various parts of the SCSI implementation. Both components are treated as "black boxes." Little information was available for explaining these observations.

Figure 3, which showed the bottleneck on a SCSI string, showed that about 25% of the available bandwidth on the SCSI string was used up by overhead associated with the SCSI protocol when four disks on the same string were accessed simultaneously. Table 11 shows a breakdown of the overheads observed for a particular trace, that of successive 32 KByte sequential read requests submitted to a single disk by a single process. The BFR for this trace was set at 1/2 (16 KBytes), and the read-ahead cache was enabled. The overheads can be attributed either to the disk or the controller, or sometimes to neither one, as in the case of an arbitration phase. Refer to Section 2.3 for a complete description of the SCSI protocol.

The trace reflects the following sequence. After receiving a command from the host, the controller arbitrates for the bus, selects the target disk, and sends the command. The disk releases the SCSI bus in order to fill its disk buffer to the amount required by the Buffer Full Ratio. When the BFR is met, the disk re-establishes communication and transfers data until its buffer empties. It then disconnects and again fills its buffer. After reconnecting and transferring the remainder of the data, the disk signals that the command is complete, and the operation ends.

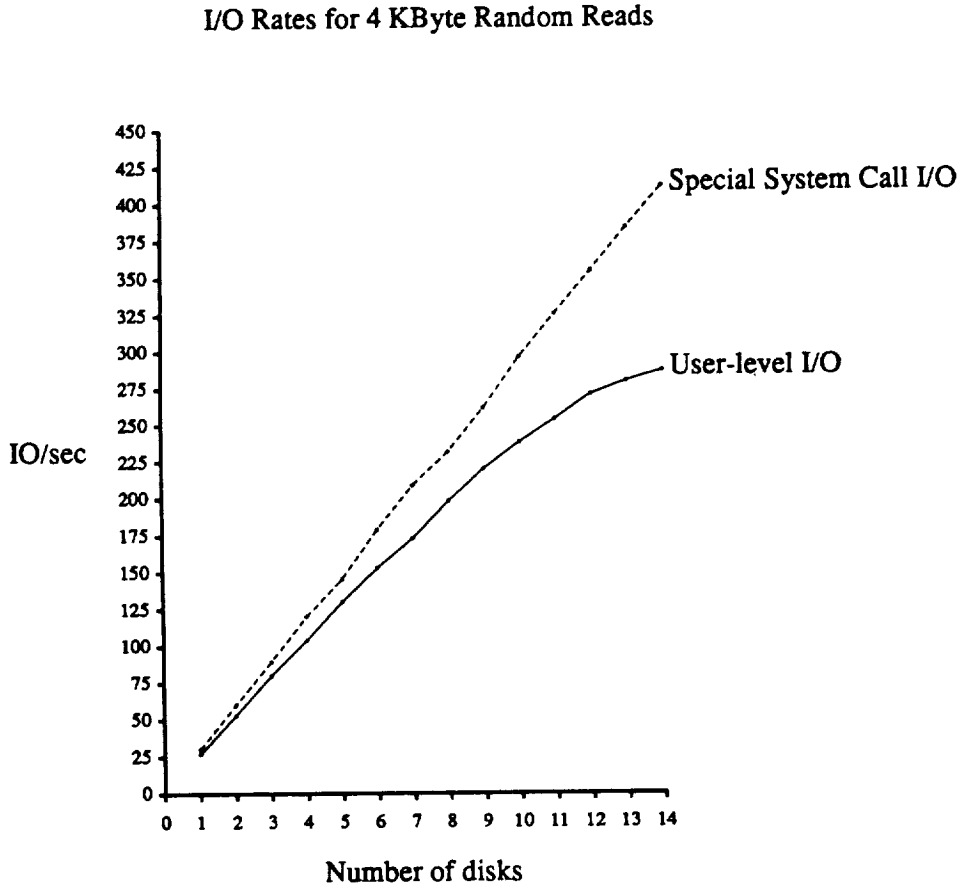


Figure 20: I/O Rates for 4 KByte random reads performed on 14 disks on four strings (each string on a separate HBA) in RAID-I. I/Os are issued from the no-copy system call in the top line, and from Sprite user level in the lower line in the graph.

Measured SCSI Overheads			
	Wren	Jaguar	neither/both
Command Setup:			
Arbitration			4 usec
Selection			3 usec
Disk Pause before request for ID message	27 usec		
Command Phase, first byte		22 usec	
Acknowledge remaining 5 command bytes		11 usec	
Disk pause before disconnect request	300 usec		
Controller acknowledge of disconnect		few usec	
Bus Free			
Rearbitration			3 usec
Reselection			3 usec
Data Transfer:			
Disk pause before request data byte	185-285 usec		
Controller ack after first 8 bytes		82 usec	
Data Transfer at 4 MB/sec			
Contr. buffer allocation after 8 KBytes		300-1700 usec	
Pause before save pointer request	200-400 usec		
Controller ack of save pointer		100 usec	
Controller ack of disconnect		30 usec	
Bus Free			
Rearbitration			3 usec
Reselection			3 usec
Disk pause before request data byte	185-285 usec		
Controller ack after first 8 bytes		82 usec	
Data Transfer at 4 MB/sec			
Contr. buffer allocation after 8 KBytes		300-1700 usec	
Command Completion:			
Disk pause before status request	500-1000 usec		
Controller ack of status		40 usec	
Controller ack of command complete		24 usec	
Total:	**1.8 msec**	**2.4 msec**	

Table 11: SCSI Overheads traced during the execution of 32 KByte sequential read operation issued using no-copy system call.

There are several notable disk overhead values. During the command setup phase, the disk pauses for 300 microseconds (usec) before requesting that a disconnect message be sent to the controller. At the beginning of the data transfer phase, the disk waits for 185-285 usec after reconnection is complete before requesting that it be allowed to send the first byte of read data to the controller. Before the second disconnect, the disk pauses for 200-400 usec before requesting that it be allowed to send a SAVE DATA POINTER message to the controller. Finally, when the operation is about to complete, the disk pauses for 500-1000 usec before entering the STATUS phase.

One of the notable controller overheads is the 82 usec pause before the controller acknowledges the first 8 bytes of the data transfer; until the eighth byte has been sent, acknowledgments are sent to the disk, but the DMA into the Jaguar's buffer space is not completely set up. Instead of going to the buffer, the first eight bytes enter a FIFO on the controller. When the FIFO fills, the controller cannot accept more data bytes until it moves the first data bytes out of the FIFO and into its local buffers.

The largest controller overhead occurs after each 8 KBytes of data have been transferred to the controller. These delays range from 300-1700 usec, and increase in size depending on the size of the transfers. They occur because the Jaguar allocates its internal data buffer space 8 KBytes at a time. (The Jaguar is designed specifically for traditional file system applications, and allocating buffer space in pieces >8 KBytes was considered to be inefficient for these applications.) Whenever an 8 KByte boundary is crossed during large transfers, another 8 KByte block of space must be allocated in the Jaguar's memory. (Note that the initial allocation for each data transfer occurs during the preceding disconnect, so the only pauses noticed are the ones that occur when an 8 KByte boundary is crossed during the transfer phase.)

One other controller overhead is notable. It takes the controller 100 usec to perform and acknowledge the SAVE DATA POINTER operation.

The overheads add up to a total of about 4.2 msec for the operation, 1.8 msec attributable to the Wren IV disk and 2.4 msec to the Jaguar HBA. The data bytes are transferred at a rate of 250 nsec per byte. At this rate, the transfer of the 32 KBytes of data requires 7.8 msec to complete. So, out of a total of 12.0 msec that the bus is busy during the operation (data transfer time plus SCSI overhead), overhead accounts for more than 1/3 of the time.

4.7 Timelines for I/O Operations

This section presents timelines for various Sprite I/O operations. These measurements were obtained by modifying the Sprite kernel to include instructions that read a high-resolution (1 usec) timer whenever certain events occured during the course of an I/O operation.

The timing intervals of interest were:

- Start System Call: For requests issued from Sprite user level, this interval is the time between the entrance to Sprite system call code and the procedure call into device driver code.
- Device Driver Time: This measurement records the time spent in the "generic" Sprite device driver before being sent to the Jaguar-specific device driver.
- Jaguar Driver Time: Records time spent in the device driver specific to the Interphase Jaguar HBA.

Start System Call	420 us
Device Driver Time	350 us
Jaguar Driver Time	380 us
Time on Jaguar Board	77900 us
Start Jaguar Interrupt Handling	67 us
DMA Free Time	9600 us
Processing before Copy	660 us
Kernel to User Copy	21000 us
Finish Processing	200 us
Total Time per Call	111000 us
Time Between Subsequent Calls	650 us

Table 12: Timeline of 128 KByte sequential read operations issued from Sprite user level.

- Time on Jaguar Board: Records the interval from the time when the SCSI command is submitted to the Jaguar board until the first instruction of the Jaguar interrupt handler is executed. The jaguar board does not interrupt the host until all disk activity is complete.
- Start Jaguar Interrupt Handling: Records the processing time in the interrupt handler up to the DMA flush.
- DMA Flush: Time spent flushing the DMA space used in the data transfer with the Jaguar.
- Processing before Copy: Processing time between DMA flush and copy operation.
- Copy Time (Kernel to User for reads, User to Kernel for writes): The copy associated with a read operation would occur at this point in the sequence. Data is DMA'd out of the Jaguar and into the kernel's address space; after the DMA operation is complete, the data must be copied from the kernel space to the user space that requested it.

  On a write operation, the copy goes from user space to kernel space, and occurs in the sequence after the "Start system call" interval, and before the "Device Driver Time" interval. After being copied to kernel space, the data is DMA'd to the Jaguar board.
- Finish Processing: This interval records the time to complete the system call execution.
- Time between Subsequent Commands: This interval records the time in Sprite between finishing execution of one system call and starting processing on the next.

Tables 12 through 21 show the timelines for operations generated as raw reads and writes from Sprite user level. These timelines show values averaged over approximately 50 operations. The tables display timelines for sequential reads and writes of size 4 KBytes, 32 KBytes and 128 KBytes, and random reads and writes of size 4 KBytes and 32 KBytes.

There are a number of interesting measurements in these figures. First, it is clear that time on the Jaguar board is the largest portion of each timeline. Most intervals on Sprite are very small by comparison. Two operations in Sprite do take a significant amount of time, as described in Section 2.4.1. They are the flushing of the cache memory used for the DMA operation between the host and a Jaguar HBA, and the copy between kernel and user space. Since these are the operations

Start System Call	250 us
User to Kernel Copy	23400 us
Before Device Driver	170 us
Device Driver Time	260 us
Jaguar Driver Time	390 us
Time on Jaguar Board	97000 us
Start Jaguar Interrupt Handling	42 us
DMA Free Time	1800 us
Finish Processing	680 us
Total Time per Call	124000 us
Time Between Subsequent Calls	190 us

Table 13: Timeline of 128 KByte sequential write operations issued from Sprite user level.

Start System Call	290 us
Device Driver Time	150 us
Jaguar Driver Time	250 us
Time on Jaguar Board	22800 us
Start Jaguar Interrupt Handling	21 us
DMA Free Time	2700 us
Processing before Copy	420 us
Kernel to User Copy	4230 us
Finish Processing	170 us
Total Time per Call	31000 us
Time Between Subsequent Calls	190 us

Table 14: Timeline of 32 KByte sequential read operations issued from Sprite user level.

Start System Call	170 us
User to Kernel Copy	4500 us
Before Device Driver	120 us
Device Driver Time	200 us
Jaguar Driver Time	270 us
Time on Jaguar Board	38200 us
Start Jaguar Interrupt Handling	26 us
DMA Free Time	570 us
Finish Processing	470 us
Total Time per Call	44500 us
Time Between Subsequent Calls	2000 us

Table 15: Timeline of 32 KByte sequential write operations issued from Sprite user level.

Start System Call	230 us
Device Driver Time	110 us
Jaguar Driver Time	270 us
Time on Jaguar Board	57600 us
Start Jaguar Interrupt Handling	32 us
DMA Free Time	2650 us
Processing before Copy	470 us
Kernel to User Copy	4450 us
Finish Processing	170 us
Total Time per Call	66000 us
Time Between Subsequent Calls	690 us

Table 16: Timeline of 32 KByte random read operations issued from Sprite user level.

Start System Call	140 us
User to Kernel Copy	4400 us
Before Device Driver	150 us
Device Driver Time	225 us
Jaguar Driver Time	260 us
Time on Jaguar Board	58000 us
Start Jaguar Interrupt Handling	15 us
DMA Free Time	590 us
Finish Processing	500 us
Total Time per Call	64300 us
Time Between Subsequent Calls	440 us

Table 17: Timeline of 32 KByte random write operations issued from Sprite user level.

Start System Call	168 us
Device Driver Time	110 us
Jaguar Driver Time	170 us
Time on Jaguar Board	4100 us
Start Jaguar Interrupt Handling	7 us
DMA Free Time	480 us
Processing before Copy	260 us
Kernel to User Copy	420 us
Finish Processing	120 us
Total Time per Call	5800 us
Time Between Subsequent Calls	90 us

Table 18: Timeline of 4 KByte sequential read operations issued from Sprite user level.

Start System Call	110 us
User to Kernel Copy	500 us
Before Device Driver	70 us
Device Driver Time	140 us
Jaguar Driver Time	170 us
Time on Jaguar Board	17800 us
Start Jaguar Interrupt Handling	20 us
DMA Free Time	145 us
Finish Processing	320 us
Total Time per Call	19300 us
Time Between Subsequent Calls	120 us

Table 19: Timeline of 4 KByte sequential write operations issued from Sprite user level.

Start System Call	140 us
Device Driver Time	140 us
Jaguar Driver Time	160 us
Time on Jaguar Board	33600 us
Start Jaguar Interrupt Handling	28 us
DMA Free Time	480 us
Processing before Copy	300 us
Kernel to User Copy	500 us
Finish Processing	85 us
Total Time per Call	35400 us
Time Between Subsequent Calls	370 us

Table 20: Timeline of 4 KByte random read operations issued from Sprite user level.

Start System Call	96 us
User to Kernel Copy	480 us
Before Device Driver	46 us
Device Driver Time	130 us
Jaguar Driver Time	190 us
Time on Jaguar Board	31700 us
Start Jaguar Interrupt Handling	18 us
DMA Free Time	240 us
Finish Processing	380 us
Total Time per Call	33300 us
Time Between Subsequent Calls	320 us

Table 21: Timeline of 4 KByte random write operations issued from Sprite user level.

that we claim tend to limit overall performance of RAID under Sprite, it is not surprising to see that they make up a significant portion of the timeline for operations issued from Sprite user level.

One interesting measurement is the speed at which the data copy occurs. The rate at which data is copied between kernel and user space varies somewhat with the size of the transfer. For 128 KByte operations, copy speed averaged 5.7 MBytes/sec. For 32 KByte operations, the copy rate was around 7 MBytes/sec, and for 4 KByte operations, the rate was around 8 MBytes/sec. The different copy rates can be explained by [Ouster2], which measured data copy operations on a Sun4; copies are performed at about 11 MBytes/sec when both the source and destination of the copy are in the cache, and at about 5 MBytes/sec when neither object is cached. Large (128 KByte) copy operations are slower because the cache itself is only 128 KBytes, so the source and destination are unlikely to be in the cache when he operation occurs. Smaller operations are more likely to have cached data, and are therefore faster.

The cache flushing rates for the DMA flush operation were around 12 MBytes/sec for read operations (where cache blocks tend to be dirty when flushed) and 50-70 MBytes/sec for write operations (where cache blocks tend to be clean when flushing occurs). This explains why the DMA flush operations in Tables 12 through 21 are four to five times longer for reads than for writes, except for the 4 KByte operations. (These small operations must perform flushes 8 KBytes at a time, which affects their flush times.)

Another observation from the graphs is that sequential write operations spend much longer on the Jaguar board than do sequential reads of the same size. The differences in the timeline tables range from 12 msec to 20 msec. The reason for this disparity is that on sequential write operations, subsequent operations suffer a missed rotation on the disk, since by the time setup of the operation is complete, the disk has spun past the point at which the previous operation stopped writing.

This observation can be used to compare the times spent on the Jaguar board by sequential and random reads and writes. Random and sequential reads of the same size differ in time spent on the Jaguar by an amount approximately equal to an average seek plus an average rotation. Random and sequential writes of the same size differ by a much smaller amount, approximately the time of an average seek; both random and sequential writes suffer a rotation penalty.

For sequential reads, where no extra seek or rotation penalty is paid, the time on the Jaguar board is proportional to the size of the transfer. Likewise, the time for copy and cache flushing is proportional to the size of the transfer.

Processing for operations other than the copy, cache flush, and time spent on the Jaguar board accounts for less than 5% of the lifetime of an operation in all the traces of Tables 12 through 21, except for the 4 KByte sequential write trace, where such processing accounts for 14% of the total.

Tables 22 through 31 shows average timelines for events of interest in operations issued by the no-copy system call. These events are:

- Time between Entry Avail and Submit Command: This rather cryptic description refers to the interval from the time that the Jaguar interrupts the host to inform it that the Jaguar is ready to accept a command for execution until the time that the command is actually submitted to the Jaguar board. The code for this processing is in the Jaguar device driver.

- Time on Jaguar Board: Time from when command is submitted until the host processor receives a completion interrupt from the Jaguar.

- Time between Interrupt from Jaguar and Entry Avail: This measures the interval between

Time between entry avail and submit command	340 us
Time on Jaguar board	91500 us
Time between interrupt and entry avail	180 us

Table 22: Timeline of 128 KByte sequential read operations issued from the special no-copy system call.

Time between entry avail and submit command	290 us
Time on Jaguar board	10800 us
Time between interrupt and entry avail	180 us

Table 23: Timeline of 128 KByte sequential write operations issued from the special no-copy system call.

the completion interrupt from the Jaguar until the time that the Jaguar is ready to accept a new command (in the Jaguar driver procedure EntryAvail).

From Tables 22 through 31, it is clear that the Jaguar board is completely dominant in these kernel traces. It is also obvious why host utilization is so low in the case of operations issued from the no-copy system call, since there is no data copy or DMA cache flush operation between subsequent operations in this case, and the processing time between successive operations is very short.

Once again, the intervals on the Jaguar board indicate that operation time for random reads and writes is roughly equal, while sequential writes suffer a penalty compared to sequential reads due to the missed rotations.

As when operations were issued from Sprite user level, for sequential reads, where no extra seek or rotation penalty is paid, the time on the Jaguar board is proportional to the size of the transfer.

5 Conclusions

The goal of RAID-I was to discover whether a disk array built from commercially available components could provide adequate performance both for traditional file system and database applications (small, random I/Os) and for large scientific and image processing applications (large, sequential I/Os). Table 32 compares the expected performance of the components of RAID-I with those actually measured. It reveals a hierarchy of bottlenecks in the system.

The most constraining performance limitation in the system is the memory bandwidth limitation of the Sun4/280 host. This limits overall sequential performance to 2.3 MBytes per second, less than can be achieved by just two Wren IV disks. CPU utilization on the Sun4/280 host limits overall small random I/O rates on the array to around 300 I/Os per second. The next level in the

Time between entry avail and submit command	120 us
Time on Jaguar board	24500 us
Time between interrupt and entry avail	57 us

Table 24: Timeline of 32 KByte sequential read operations issued from the special no-copy system call.

Time between entry avail and submit command	200 us
Time on Jaguar board	40400 us
Time between interrupt and entry avail	130 us

Table 25: Timeline of 32 KByte sequential write operations issued from the special no-copy system call.

Time between entry avail and submit command	130 us
Time on Jaguar board	57000 us
Time between interrupt and entry avail	54 us

Table 26: Timeline of 32 KByte random read operations issued from the special no-copy system call.

Time between entry avail and submit command	442 us
Time on Jaguar board	55500 us
Time between interrupt and entry avail	100 us

Table 27: Timeline of 32 KByte random write operations issued from the special no-copy system call.

Time between entry avail and submit command	140 us
Time on Jaguar board	3200 us
Time between interrupt and entry avail	43 us

Table 28: Timeline of 4 KByte sequential read operations issued from the special no-copy system call.

Time between entry avail and submit command	120 us
Time on Jaguar board	19400 us
Time between interrupt and entry avail	46 us

Table 29: Timeline of 4 KByte sequential writes operations issued from the special no-copy system call.

Time between entry avail and submit command	120 us
Time on Jaguar board	33200 us
Time between interrupt and entry avail	52 us

Table 30: Timeline of 4 KByte random read operations issued from the special no-copy system call.

Time between entry avail and submit command	150 us
Time on Jaguar board	32500 us
Time between interrupt and entry avail	60 us

Table 31: Timeline of 4 KByte random write operations issued from the special no-copy system call.

bottleneck hierarchy is the VME backplane limitation. One of the big surprises of this study is that the VME backplane limited bandwidth for operations using the no-copy system call to only 7.5 MBytes/sec in Figure 19. (The highest bandwidth ever measured on RAID-I was 9 MBytes/sec for 128 KBytes sequential reads.) This VME limit is significantly lower than the expected 15 MByte/sec limit. The next observed performance limitation is that of the Jaguar HBAs, which are only capable of supporting 4 MBytes/sec from the two strings they control. If the other bottlenecks of RAID-I were eliminated, the four HBAs in the system would limit performance to 16 MBytes/sec. SCSI overheads limit bandwidth on a string to 3 MBytes/sec. The last performance limitation in the system is the amount of data that can be delivered by the disks; I measured the expected performance of 1.3 MBytes/sec for large operations and 30 I/Os per second for small operations on the disks.

These results suggest a number of conclusions. First, a disk array built from off-the-shelf parts has achieved reasonably good performance for small random I/O operations, the standard design point for today's systems. However, the array is not adequate for providing high throughput for large sequential operations. To get better performance out of an array like RAID-I, we need a more powerful host CPU, a more powerful CPU on the controller (to handle the SCSI protocol more quickly), and more bandwidth available on the controller. (The Jaguar can only absorb half the potential bandwidth of the two strings attached to it.) The off-the-shelf parts used in the construction of RAID-I were not designed with the large, sequential workloads that we used to test RAID-I in mind. Rather, they were designed for traditional small random accesses, and they do provide adequate performance for such applications. In our second RAID prototype, which diverges substantially from the simple design of RAID-I, we hope to build controller boards that can support the bandwidth of all the disks attached to them, so that we can realize the potential bandwidth of the array.

The need for greater memory bandwidth on the host CPU deserves special consideration. This was the single most important factor limiting the performance of the disk array, and it is not a problem unique to the Sun4/280. Many of the fastest workstations currently being produced have limited memory system bandwidth. The RAID group is experiencing difficulting finding a machine with adequate memory bandwidth to support the needs of our second prototype machine. Disk arrays will not be able to deliver bandwidth to the CPU unless the memory system is capable of consuming what the array delivers.

Sprite, with its long copy overheads and traditional file system style of accessing data 4 KBytes at a time, is not particularly suitable for achieving very high bandwidth on the array when operations are issued to raw devices from user level, as was done in most of the tests in this study. In addition, when a RAID driver is used to access the disks, there will be additional overhead, since in that case, small writes do parity updates that result in additional disk accesses. In order to improve the performance of such writes, we plan to use the Log Structured File System being developed at U.C. Berkeley, which turns all small write operations into large ones [Rosenblum].

Operating systems should also support asynchronous I/O. This study showed a fundamental performance difference between large sequential reads and writes. Write performance for a disk is lower than read performance because disks appear to suffer a missed rotation on writes. Being able to issue subsequent write requests before the current operation completes would allow this missed rotation to be avoided. Asynchronous I/O is the key to improving sequential write performance.

A final conclusion of this work is that measuring the performance and attributing the overheads of off-the-shelf hardware is a daunting task. One of the reasons that RAID-II will not be built

Component	I/O Rates (I/Os per sec)			Bandwidth (MBytes/sec)		
	Expected	No-Copy	Full Sprite	Expected	No-Copy	Full Sprite
Single disk	25-30	29	27	1.3	1.3	1.3
String (4 disks)	100-120	114	103	4	3	2.3
Overall (14 disks)	320	320	270			
Overall (13 disks)				10-15	7.5	2.3
Best Ever (16 disks)				10-15	9	

Table 32: Expected versus Actual performance of RAID-I for operations issued from the No-Copy special system call and from Sprite user level. I/O rates were measured for 4 KByte random read operations. Bandwidth was measured for 32 KBytes sequential read operations. Bandwidth for the "Best Ever" case was for 128 KByte sequential reads. Note that the "expected" 10-15 MByte/sec bandwidth for the array is the expected limit of the VME backplane.

exclusively from off-the-shelf parts is so that hardware and software instrumentation can be added to the system to make performance measurements easier.

6 Acknowledgements

I am grateful to Richard Drewes and Mendel Rosenblum for their help over the life of this project. Thanks also to Randy Katz, Ken Lutz, Mary Baker, Peter Chen, Ed Lee, Garth Gibson and Ethan Miller for all their assistance and advice.

References

[Ancot] *SCSI Bus Analyzer/Emulator Model DCS-202 User's Manual*, ANCOT Corporation, Redwood City, CA.

[Lee] Ed Lee, "Software and Implementation Issues in the Implementation of a RAID Prototype ", Masters report, U.C. Berkeley, May, 1990.

[Jaguar] *V/SCSI 4210 Jaguar High Performance VMEbus Dual SCSI Host Adaptor-User's Guide*,Interphase Corporation, Dallas, TX.

[Ouster] John K. Ousterhout, Herve Da Costa, et. al., "A Trace-Driven Analysis of the UNIX 4.2 BSD File System", Proceedings of the 10th SOSP, *Operating Systems Review*, Vol. 19, No. 5, December 1985, pp. 15-24.

[Ouster2] John K. Ousterhout, "Why Aren't Operating Systems Getting Faster As Fast as Hardware?", USENIX Summer Conference, Anaheim, CA, June, 1990.

[RAID] Dave Patterson, et. al, "A Case for Redundant Arrays of Inexpensive Disks", ACM SIGMOD, Chicago, IL, June, 1988.

[Rosenblum] Mendel Rosenblum, "The LFS Storage Manager", USENIX Summer Conference, Anaheim, CA, June, 1990.

[SCSI] *SCSI Guidebook*, Adaptive Data Systems, Inc., Pomona, CA, 1985.

[Wren] *Product Specification for Wren IV SCSI Model 94171-344*, Control Data Corporation, Minneapolis, MN.

[ZBR] "Disk Drive Capacity and Performance Increases Through ZBR Variable Track Capacity Recording," Al U. Sharon, SDNC 1989 Proceedings, May 23-25, 1989, Santa Clara, CA.