References

Denning, P. J. 1970. Virtual memory. *Comput. Surveys* 2(3):153–170.

Dorf, R. C. 1992. *The Electrical Engineering Handbook*, 1st ed. CRC Press, Inc., Boca Raton, FL.

Engler, D. R., Kaashoek, M. F., and O'Toole, J., Jr. 1995. Exokernel: an operating system architecture for application-level resource management, pp. 251–266. In *Proc. 15th Symp. on Operating Systems Principles*.

Hennessy, J. L. and Patterson, D. A. 1990. *Computer Architecture: A Quantitative Approach*, 1st ed. Morgan Kaufmann, San Mateo, CA.

Hill, M. D. 1988. A case for direct-mapped caches. *IEEE Comput*. 21(12):25–40.

IEEE Computer Society. 1993. *IEEE Standard for High-Bandwidth Memory Interface Based on SCI Signaling Technology (RamLink)*. Draft 1.00 IEEE P1596.4-199X.

Jouppi, N. 1990. Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers, pp. 364–373. In *Proc. 17th Annual Int. Symp. on Computer Architecture*.

Kogge, P. M. 1981. *The Architecture of Pipelined Computers*, 1st ed. McGraw–Hill, New York.

Kroft, D. 1981. Lockup-free instruction fetch/prefetch cache organization, pp. 81–87. In *Proc. 8th Annual Int. Symp. on Computer Architecture*.

Lam, M. S., Rothberg, E. E., and Wolf, M. E. 1991. The cache performance and optimizations of blocked algorithms, pp. 63–74. In *Proc. 4th Annual Symp. on Architectural Support for Programming Languages and Operating Systems*.

Mowry, T. C., Lam, M. S., and Gupta, A. 1992. Design and evaluation of a compiler algorithm for prefetching, pp. 62–73. In *Proc. 5th Annual Symp. on Architectural Support for Programming Languages and Operating Systems*.

Rambus 1992. *Rambus Architectural Overview*. Rambus Inc., Mountain View, CA.

Seznec, A. 1993. A case for two-way skewed-associative caches, pp. 169–178. In *Proc. 20th International Symposium on Computer Architecture*.

Smith, A. J. 1986. Bibliography and readings on CPU cache memories and related topics. *ACM SIGARCH Comput. Architecture News* 14(1):22–42.

Smith, A. J. 1991. Second bibliography on cache memories. *ACM SIGARCH Comput. Architecture News* 19(4):154–182.

Talluri, M. and Hill, M. D. 1994. Surpassing the TLB performance of superpages with less operating system support, pp. 171–182. In *Proc. 6th Int. Symp. on Architectural Support for Programming Languages and Operating Systems*.

Further Information

Some general information on the design of memory systems is available in *High-Speed Memory Systems* by A. V. Pohm and O. P. Agarwal. 1983. Reston Publishing, Reston, VA.

Computer Architecture: A Quantitative Approach by John Hennessy and David Patterson [Hennessy and Patterson 1990] contains a detailed discussion on the interaction between memory systems and computer architecture.

For information on memory system research, the recent proceedings of the *International Symposium on Computer Architecture* contain annual research papers in computer architecture, many of which focus on the memory system. To obtain copies, contact the IEEE Computer Society Press, 10662 Los Vaqueros Circle, P.O. Box 3014, Los Alamitos, CA 90720-1264.

19 Buses

19.1 Introduction
19.2 Bus Physics
Transmission-Line Concepts • Signal Reflections • Wire-OR Glitches • Signal Skew • Cross-Coupling Effects
19.3 Bus Arbitration
Centralized Arbitration • Decentralized Arbitration
19.4 Bus Protocol
Asynchronous Protocol • Synchronous Protocol • Split-Transaction Protocol
19.5 Issues in SMP System Buses
Cache Coherence Protocols • Bus Arbitration • Bus Bandwidth • Memory Access Latency • Synchronization and Locking
19.6 Putting It All Together — CCL-XMP System Bus
19.7 Historical Perspective and Research Issues

Windsor W. Hsu
IBM Research

Jih-Kwon Peir
University of Florida

19.1 Introduction

The *bus* is the most popular communication pathway among the various components of a computer system. The distinguishing feature of the bus is that it consists of a single set of shared communication links to which many components can be attached. The bus is not only a very cost-effective means of connecting various components together, but also is very versatile in that new components can be added easily. Furthermore, the bus has a broadcasting capability which can be extremely useful. The downside of the shared communication links is that they allow only one communication to occur at a time and the bandwidth does not scale with the number of components attached. Nevertheless, the bus is very popular because there are many situations where several components need to be connected together but they need not all transmit at the same time. This kind of requirement maps naturally onto a bus, allowing a very cost-effective solution. However, there are cases where the bus does become a communication bottleneck. In such cases, very aggressive bus designs have been attempted, but there comes a point where the fundamental characteristic of the bus cannot be overcome and more expensive solutions such as point-to-point links have to be used.

Buses are used at every level in the computer system. For instance, within the processor itself, the bus is often the means of communication between the register file and the various execution units. At a higher level, the processor is connected to the memory subsystem through the *system bus*. Today's computers typically have a fast peripheral bus called a *local bus* which directly interfaces onto the system bus to provide a high bandwidth for demanding devices such as the graphics adaptor. Other less demanding peripheral devices are attached to the *I/O bus*. In the old days, the processor, memory subsystem, and I/O devices were all plugged onto the *backplane bus,* which is so called because the bus runs physically

along the backplane of the computer chassis. The various buses are each optimized for a particular set of performance requirements and cost constraints and may thus seem very different from one another. However, their underlying issues are fundamentally the same.

A major requirement for designing a bus or simply comprehending a bus design is a proper understanding of the electrical and mechanical behavior of the bus. As buses are pushed to provide higher data rates, physical phenomena such as signal reflection, crosstalk, skew, etc., are becoming more significant and have to be handled carefully. Because the communication medium of a bus is shared by multiple devices, at most one transmission can be initiated at any time by any device. The other devices can act only as receivers or **bus slaves** for the transmission. A device that is capable of initiating and controlling a communication is called a **bus master**. In order to ensure that only one bus master is talking at any one time, the bus masters have to go through a **bus arbitration** process before they can gain control of the bus. Once a bus master has been granted control of the bus, a **bus protocol** has to be followed by the master and the slave in order for them to understand one another. The specifics of the protocol can vary widely, depending on the functional and performance requirements. For instance, the protocol used in the system bus of a uniprocessor is dramatically different from that used in the system bus of a shared-memory **symmetric multiprocessor** (SMP).

In this chapter, an overview of the basic underlying physics of computer buses is presented first. This is followed by a discussion of important issues in bus designs, including bus arbitration and various communication protocols. Because of the increasing prevalence of SMP systems and the special challenges they pose for buses, a separate section is devoted to discussing special bus issues in such machines. The discussion is wrapped up with a case study of a modern SMP system bus design. Finally, a historical perspective of computer buses and the related research issues are given in the last section.

19.2 Bus Physics

Computer buses are becoming wider and are being run at higher frequencies in order to keep up with the phenomenal improvement in CPU performance. As the physical and temporal margins in bus designs are reduced, it is imperative that electrical phenomena such as signal reflections, skew, crosstalk, etc., be understood and properly handled. In this section, we introduce the basic ideas behind these phenomena. A more detailed discussion can be found in Giacomo [1990].

19.2.1 Transmission-Line Concepts

Electrical signals propagate with a finite speed that depends on the propagation medium. For instance, it takes approximately 5 ns for an electrical signal to travel 1 m along a typical copper wire. However, when analyzing electrical circuits, we often ignore their spatial properties. For example, we are seldom concerned with where each element of the circuit is located. We can do this because the circuits we usually encounter are lumped. In other words, their physical dimensions are small enough that for their particular applications, electromagnetic waves propagate across the circuits virtually instantaneously. However, this is not the case with high-speed buses. In this subsection, we introduce the basic concepts of the transmission-line model which are needed to understand the electrical behavior of today's buses.

In general, a connection is considered a transmission line if the propagation time of an electrical signal through it is a significant part of the rise time, fall time, or width of the signal. A good rule of thumb is to consider anything above $\frac{1}{4}$ as a "significant part." A transmission line has a characteristic impedance Z_0. If the line impedance Z_L changes as a signal propagates down the line, part of the power travels backwards as a reflected signal. The reflection coefficient is given by $\Gamma = (Z_L - Z_0)/(Z_L + Z_0)$. In other words, the strength of the reflected signal increases with the magnitude of the impedance mismatch.

Consider the circuit in Figure 19.1. It contains a voltage source V_s with internal resistance R_s connected by a pair of transmission lines to a load of resistance Z_L. We assume that it takes T units of time for an electrical signal to propagate one-way across the circuit. At time $t = 0$, the switch is closed and a voltage

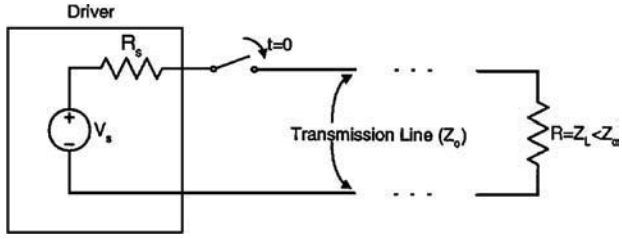

FIGURE 19.1 Circuit for transmission-line example.

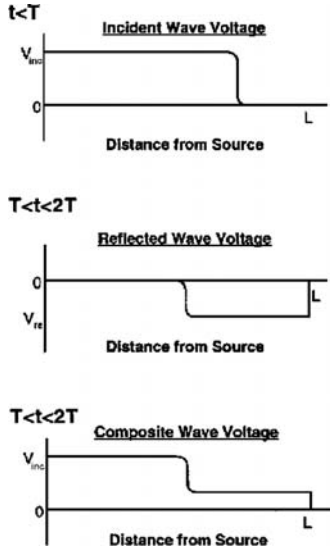

FIGURE 19.2 Transmission-line voltage waveform.

pulse is sent down the transmission lines toward the load. Figure 19.2 contains a short chronicle of the voltage waveform seen on the lines. For time $t < T$, the signal sees only Z_0, the impedance of the line. Thus, the voltage of the incident wave $V_{\text{inc}} = V_s Z_0/(R_s + Z_0)$. When the incident wave hits the load at time $t = T$, it sees an additional impedance Z_L which reflects the incident wave with a coefficient of $\Gamma = (Z_L - Z_0)/(Z_L + Z_0)$. In Figure 19.2, we assume that $Z_L < Z_0$ so that the reflected wave is negative with voltage $V_{\text{re}} = \Gamma V_{\text{inc}}$. If the transmission lines are not properly terminated at the driver, i.e., $R_s \neq Z_0$, there will be additional reflections before the system settles down. The composite signal at any point along the line is the instantaneous sum of all the incident and reflected signals. Note that all the signals are subject to line losses, especially in their high-frequency components. Thus, pulses eventually lose their shape.

When the circuit is in steady state, the above analysis should agree with the lumped circuit model. For simplicity, let us assume that the transmission lines are properly terminated at the driver, i.e., $R_s = Z_0$. In this case, there is only one reflection and the circuit settles down at time $t = 2T$. Thus, we would expect the steady-state voltage to be the sum of V_{inc} and V_{re}. This works out to $V_s Z_L/(Z_L + Z_0)$, a result which agrees with that predicted by the lumped-circuit model. An intuitive way to reason about transmission lines is to think in terms of feedback. In the beginning, the signal has no clear idea of how much impedance it will encounter in the circuit. Thus, it makes a guess based on the impedance it has already seen. As the signal propagates, it learns more about the circuit, and this information is fed back in the form of reflections, so that eventually Kirchhoff's circuit laws are satisfied.

19.2.2 Signal Reflections

The consequence of having signal reflections in the system is that glitches and extra pulses may appear on the bus. This may cause some unexpected and obscure problems. Some of the more common symptoms of reflection problems include the following:

- A board that stops working after another board is plugged into the system.
- A board that works only in a particular slot on the bus.
- A system that works only when the boards are arranged in a specific order.

Reflections are typically most significant at the various sources and loads on the bus. We can reduce the magnitude of these reflections by matching the impedance of the sources and loads to that of the lines.

This can be accomplished by adding a series or parallel resistance or by using a clamping diode. Impedance matching is complicated by the fact that the properties of bus drivers change as they switch on or off. For better impedance matching, small voltage swings and low-capacitance drivers and receivers are helpful. Note that reflections can also occur at other impedance discontinuities such as interboard connections, board layer changes, etc. To accurately model all these effects, computer simulations using tools such as SPICE are often needed.

19.2.3 Wire-OR Glitches

Wire-OR logic is a kind of logic where the outputs of several open-collector gates are connected together in such a way as to realize a logical *OR* function. A sample circuit is shown in Figure 19.3. Notice that the voltage on the line is low as long as any one of the transistors is turned on. Thus this circuit implements the logical *NOR* function or the *OR* function with the output asserted low. Wire-OR is very useful in bus arbitration. For instance, it enables the system to determine whether any bus master wishes to use the bus. Most buses use wire-OR logic for at least a few lines.

However, wire-OR lines are subject to a fundamental phenomenon known as *wire-OR glitch*. During an active to high-impedance transition, a glitch of up to one round-trip delay in width may appear on a wire-OR line. This phenomenon is a result of the finite propagation speed of electrical signals on a transmission line. Consider the case where only transistors 1 and *n* are initially turned on. Suppose that transistor *n* is now turned off. The current that it was previously sinking continues to flow, thus creating a signal which propagates along the line.

A more detailed explanation of the wire-OR glitch is given in Gustavson and Theus [1983]. Various ways of dealing with it are discussed in Gustavson and Theus [1983] and Taub [1983a, 1983b]. In the IEEE Futurebus, the wire-OR glitch problem is mitigated by the fact that the bus specification imposes constraints on when devices can switch on or off, effectively setting a limit on the maximum glitch duration [Taub 1984].

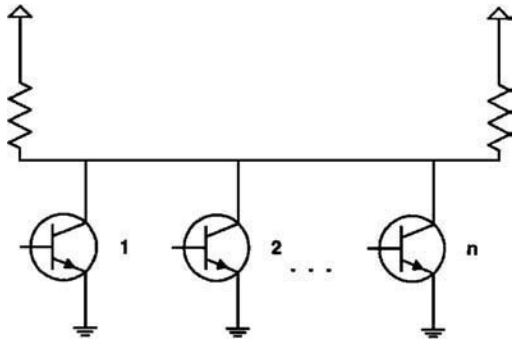

FIGURE 19.3 Wire-OR circuit.

19.2.4 Signal Skew

Another important electrical phenomenon in buses is *signal skew*. Because of differences in transmission lines, loading, etc., slight differences in the propagation delay of different bits in a word are inevitable. These differences are known as signal skew.

In a transmission, the receiver must be able to somehow determine when all the bits of a word have arrived and can be sampled. The effect of skew is to reduce the window during which the receiver can assume that the data are valid. This effectively limits the data rate of the bus. A wide bus consists of more parallel lines and is thus subject to more skew. In general, skew can be reduced by paying meticulous attention to the impedance of the bus lines. Synchronous buses have to deal with the additional problem of clock to data skew. An approach that has been taken to minimize this skew is to loop back the clock line at the end of the bus. When doing a data transfer, the clock signal that propagates in the same direction as the data transfer is used. Rambus uses this technique to minimize skew with respect to its aggressive 250-MHz clock [Rambus 1992].

19.2.5 Cross-Coupling Effects

A signal-carrying line sets up electrostatic and magnetic fields around it. In a bus, the lines run parallel and close to one another. Thus the fields from nearby lines intersect, causing a signal on one line to affect the signal on another. This is called crosstalk or coupling noise.

A simple way to reduce this effect is to spatially separate the bus lines so that the fields do not interfere with one another. However there is clearly a limit to how far we can carry this. Another way to reduce both the mutual capacitance and inductance of the lines is to introduce ground planes or wires near the bus lines. But this has undesirable side effects such as increasing the self-capacitance of the lines. An approach commonly taken to reduce coupling effects is to separate the lines with an insulator that has a low dielectric constant. Typically, combinations of these techniques are used in a bus design.

19.3 Bus Arbitration

Buses are ubiquitous in computer systems because they are a cost-effective and versatile means of connecting several devices together. The cost-effectiveness and versatility of buses stems from the fact that a bus has only one communication medium, which is shared by all the devices. In other words, at most one communication can occur on a bus at any one time. This implies that there must be some mechanism to decide which bus master has control of the bus at a given time. The process of arbitrating between requests for bus control is called bus arbitration. Bus arbitration can be handled in different ways depending on the performance requirements and cost constraints.

19.3.1 Centralized Arbitration

In *centralized arbitration*, there is a special device, the central arbiter, which is in charge of granting bus control to one of the bus masters. The fact that there is only one arbiter means that the centralized scheme has a single point of failure. In general, centralized arbitration can be further divided into two schemes, depending on how the bus masters are connected to the arbiter.

In the first scheme, the central arbiter is connected to each of the bus masters through private two-way connections. Because the bus requests can be made independently and in parallel by the bus masters, this is sometimes known as *centralized independent requests arbitration* or *centralized parallel arbitration*. Notice that the connections in this system form star networks emanating from the central arbiter. One such network carries the bus request signals from the bus masters to the arbiter. Another carries the bus grant signal from the arbiter back to one of the bus masters. Various arbitration policies can be implemented in the arbiter, making this a very flexible scheme. This scheme is fast because there are direct connections between any bus master and the arbiter. However, all these direct connections lead to a high implementation

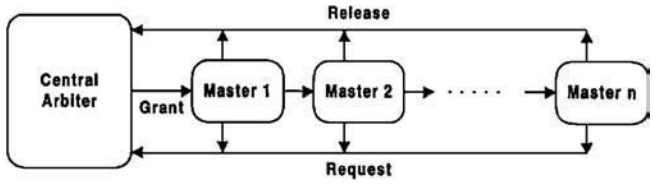

FIGURE 19.4 Daisy-chain bus arbitration.

cost. Furthermore, the use of direct connections means that the arbitration signals do not appear on the bus. This makes bus monitoring for debugging and diagnostic purposes difficult.

The second centralized arbitration scheme is known as *centralized serial priority arbitration*. In this scheme, there is a single bus grant signal line, which is routed through each of the bus masters as shown in Figure 19.4. This form of connection is known as a daisy chain. Hence, centralized serial priority arbitration is more commonly referred to as *daisy-chain arbitration*. In this scheme, there is a common wire-OR bus request line. A bus master may take control of the bus only if it has made a request and its incoming grant line is asserted. A bus master that does not wish to use the bus is required to forward the bus grant signal along the daisy chain. Notice that this implies an implicit priority assignment — the nearer a bus master is to the arbiter, the higher is its priority. The main advantage of daisy-chain arbitration is that it requires very few interconnections and the interface logic is simple. However, bus allocation in this scheme is slow because the grant signal has to travel along the daisy chain. Furthermore, the implicit priority scheduling may cause low-priority requests to be locked out indefinitely. Finally, as in centralized parallel arbitration, daisy-chain arbitration does not facilitate debugging and diagnosis. The VMEbus uses a variation of this scheme with four daisy-chained grant lines, which enable it to implement a variety of scheduling algorithms [Giacomo 1990].

19.3.2 Decentralized Arbitration

In *decentralized arbitration*, each bus master has its own arbitration and allocation logic. The responsibility of deciding who has control of the bus is distributed among the bus masters. Thus, this scheme is also known as *distributed arbitration*.

Typically, the bus contains n arbitration wire-OR lines and each bus master is assigned a unique n-bit arbitration number according to some priority scheme. During arbitration, if a master wishes to use the bus, it drives the arbitration lines with its arbitration number. If a master detects that the arbitration lines are carrying a higher priority number, it stops driving the less significant bits of its number. When the system settles down, the arbitration lines will indicate the winner, which is the participating master with the highest priority. The time for the system to settle down can be specified as a fixed time in the bus protocol. An alternative is to use an additional wire-OR line to indicate whether competing bus masters have completed arbitration. As with any priority scheme, the possibility of access starvation of the low-priority masters has to be considered. In the IEEE Futurebus, bus masters are divided into priority and fairness modules depending on whether they have any particularly urgent needs such as having to meet real-time constraints [Taub 1984]. A priority module can issue bus requests whenever it needs to. A fairness module, after winning an arbitration, will have to wait for all pending requests to be serviced before it can issue another request [Taub 1984]. Note that this does not guarantee but only helps to ensure that every module will be able to get a portion of the bus bandwidth.

The major disadvantage of the distributed scheme is that it requires relatively complex arbitration logic in each bus master and several arbitration lines on the bus. However, this scheme allows very fast bus allocation and a flexible assignment of priorities to the bus masters. Distributed arbitration is also more fault-tolerant than the centralized schemes in that the failure of a single bus master does not necessarily affect the operation of the bus. Variations of this scheme are widely used in buses such as the IEEE Futurebus, Nubus, Multibus II, Fastbus [Borrill 1985], and the Powerpath-2 System Bus of the SGI Challenge [Galles and Williams 1994].

The scheme described above is sometimes known as *distributed arbitration by self-selection* because each master decides whether it has won the race, effectively letting the winner select itself. Some computer networks, such as Ethernet, use another form of distributed arbitration which relies on collision detection [Metcalfe and Boggs 1976].

19.4 Bus Protocol

The bus is a shared resource. In order for it to function properly, all the devices on it must cooperate and adhere to a protocol or set of rules. This protocol defines precisely the bus signals that have to be asserted by the master and slave devices in each phase of the bus operation. In this section, we discuss some of the key options in designing bus protocols.

19.4.1 Asynchronous Protocol

In a communication, the sender and receiver must be coordinated so that the sender knows when to talk and the receiver knows when to listen. There are two basic ways to achieve proper coordination. This subsection discusses the asynchronous protocol. The next describes the synchronous design.

An asynchronous system does not have an explicit clock signal to coordinate the sender and receiver. Instead a *handshaking protocol* is used. In a handshaking protocol, the sender and receiver can proceed to the next step in the bus operation only if both of them are ready. In other words, both parties have to shake hands and agree to proceed before they can do so.

Figure 19.5 shows a basic handshaking protocol. Assume that *ReadReq* is used to request a read from memory, *DataReady* is used to indicate that data are ready on the data lines, and *Ack* is used to acknowledge the *ReadReq* or *DataReady* signals of the other party. Suppose a device wishes to initiate a memory read. When memory sees the *ReadReq*, it reads the address on the address bus and raises the *Ack* in acknowledgment of the request (step 1). When the device sees the *Ack*, it deasserts *ReadReq* and stops driving the address bus (step 2). Once memory sees that *ReadReq* has been deasserted, it drops the *Ack* line to acknowledge that it has seen the *ReadReq* signal (step 3). A similar exchange is carried out when memory is ready with the data that has been requested (steps 5–7). Notice that the two parties involved in the protocol take turns to respond to one another. They proceed in lockstep. This is how the handshaking protocol is able to coordinate the two devices.

The handshaking protocol is relatively insensitive to noise because of its self-timing nature. This self-timing nature also allows data to be transferred between devices of any speed, giving asynchronous designs the flexibility to handle new and faster devices as they appear. Thus, asynchronous designs are better able to scale with technology improvements. Furthermore, because clock skew is not an issue, asynchronous buses can be physically longer than their synchronous counterparts. The disadvantage of asynchronous designs lies in the fact that the handshaking protocol adds significant overhead to each data transfer and is thus slower than a synchronous protocol. As in any communication between parties with different clocks, there

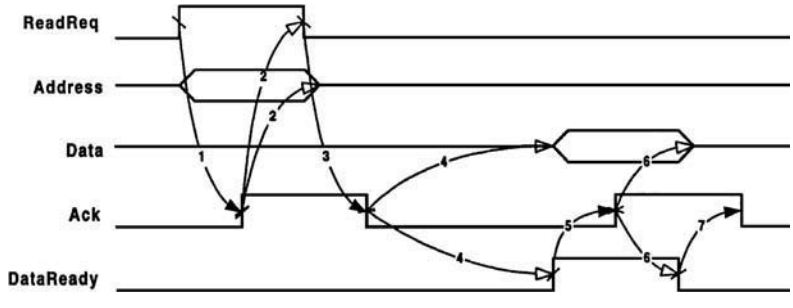

FIGURE 19.5 A basic handshaking protocol.

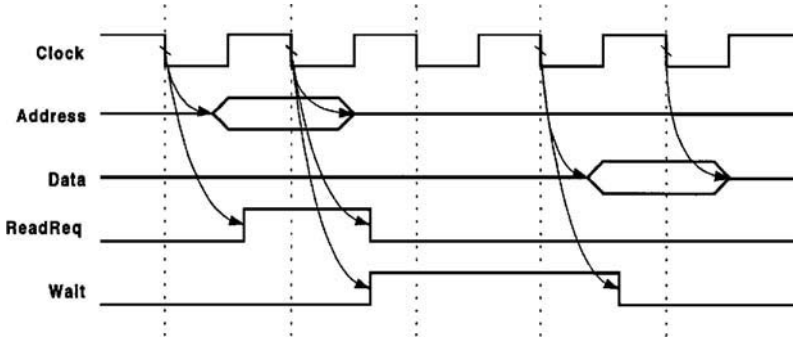

FIGURE 19.6 A basic synchronous protocol.

is an additional problem of synchronization failure when an asynchronous signal is sampled. Asynchronous designs are typically used when there is a need to accommodate many devices with a wide performance range and when the ability to incrementally upgrade the system is important. Thus, many of the asynchronous buses such as VMEbus, Futurebus, MCA, and IPI are backplane or I/O buses [Giacomo 1990].

19.4.2 Synchronous Protocol

In synchronous buses, the coordination of devices on the bus is achieved by distributing an explicit clock signal throughout the system. This clock signal is used as a reference to determine when the various bus signals can be assumed to be valid. Figure 19.6 shows a basic synchronous protocol coordinating a memory read transaction. Notice that all the signal changes happen with respect to the clock. In this particular example, the system is negative-edge triggered, which means that the signals are sampled on the falling edge of the clock.

An important design decision in synchronous buses is the choice of the clock frequency. Once a frequency is selected, it becomes locked into the protocol. The clock frequency must be chosen to allow sufficient time for the signals to propagate and settle throughout the system. Allowances must also be made for clock skew. Thus, the clock frequency is limited by the length of the bus and the speed of the interface logic. All things being equal, shorter buses can be designed to run at higher speeds.

The main advantage of the synchronous protocol is that it is fast. It also requires relatively few bus lines and simple interface logic, making it easy to implement and test. However, the synchronous protocol is less flexible than the asynchronous protocol in that it requires all the devices to support the same clock rate. Furthermore, this clock rate is fixed and cannot be raised compatibly to take advantage of technological advances. In addition, the length of synchronous buses is limited by the difficulty of distributing the clock signal to all the devices at the same time. Synchronous buses are typically used where there is a need to connect a small number of very tightly coupled devices and where speed is of paramount importance. Thus, synchronous buses are often used to connect the processor and the memory subsystem.

19.4.3 Split-Transaction Protocol

In order to increase the effective bandwidth of a bus, a **split-transaction protocol** can be used. The basic observation behind this protocol is that the bus is not being used to transmit information throughout the entire duration of a transaction, but only at the start and toward the end of the transaction. The idea is thus to split a bus transaction into a request transaction and a reply transaction so as to allow the bus to be released for other uses in between the request and reply stages. Figure 19.7 illustrates the idea. Clearly, this protocol only makes sense when the system has more than one bus master and the memory system is sophisticated enough to handle multiple overlapping transactions. This protocol is sometimes also known as a *connect/disconnect protocol*, *pipelined protocol*, or *packet-switched protocol.*

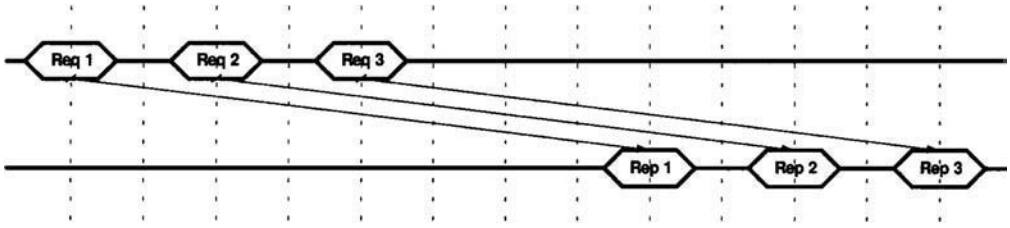

FIGURE 19.7 A split-transaction protocol.

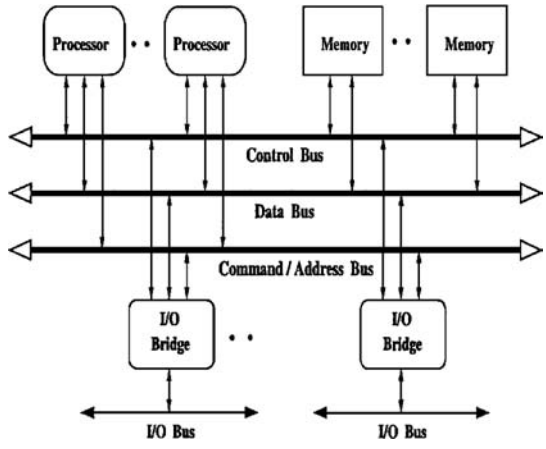

FIGURE 19.8 A typical multiprocessor system bus.

Although the split-transaction protocol allows more efficient utilization of bus bandwidth than a protocol that holds on to the bus for the whole transaction, it usually has a higher latency because the bus has to be acquired twice — once for the request and once for the reply. Furthermore, the split-transaction protocol is expensive to implement because it requires that the bus transactions be tagged and tracked by each device. Split-transaction protocols are widely used in the system buses of shared-memory SMPs, because bus bandwidth is a big issue in these machines.

Notice from Figure 19.7 that even with split transactions, the bus bandwidth is not totally utilized. This is because some bus cycles are needed to acquire the bus and to set up the transfer. Furthermore, some buses require a cycle of turnaround time between different masters driving the bus. Another way to increase effective bus bandwidth is thus to amortize this fixed cost over several words by allowing the bus to transfer multiple contiguous words back to back in one transaction. This is known as a *burst protocol* or *block transfer protocol* because a contiguous block of several words is transferred in each transaction.

19.5 Issues in SMP System Buses

Microprocessor-based symmetric multiprocessors (SMPs) with shared snooping buses have become an industry standard for building midrange departmental servers [Peir et al. 1993, Galles and Williams 1994]. In this section, we devote special attention to the bus issues that arise in SMP designs.

A typical SMP bus architecture (usually referred to as a *SMP system bus*, or simply *system bus*) is illustrated in Figure 19.8. SMP system buses require high bandwidth and low latency to connect multiple processors, memory modules, and I/O bridges. A system bus is composed of independent signal lines

for transmitting control, command/address, and data information. These lines can be grouped into what is commonly called the *control bus*, the *command/address bus,* and the *data bus*, respectively. Each bus can be acquired and used independently. Some of the control lines, such as the bus request signal to the arbiter, are point-to-point connections; therefore, they can be used before the bus is acquired. Also, the wire-OR control lines need not be acquired before they are used. Note also that the signal lines for sending the command and the address are considered as the same bus because they are always acquired together.

In general, each system bus request traverses through a number of stages, each of which may take one or more bus cycles.

- *Bus arbitration:* The requesting processor needs to go through an arbitration process in order to gain access to the shared system bus.
- *Command issuing:* After winning the bus arbitration, the processor issues the command along with an address on the command/address bus. Certain requests, e.g., a cache line writeback, also require access to the data bus in this stage.
- *Cache snooping:* Once a valid command is received, all the system bus masters (processors, I/O bridges) search their own cache directory and initiate proper cache coherence activities to maintain data coherence among multiple caches. This is a unique requirement for SMP system buses. A description of **cache coherence protocols** is given in Cache Coherence Protocols subsection of this section.
- *Acknowledgment:* The snoop results are driven onto the bus. The issuing processor has to update its cache directory based on the results.
- *Data transfer:* When a bus request incurs a data transfer, such as a line-fill request issued upon a cache miss, the transfer of data is carried out at this stage through the data bus.
- *Completion:* The bus transaction is completed.

Several important and unique issues in SMP system bus design are outlined as follows.

1. *Bus physics:* Due to the high bandwidth requirement, SMP system buses face the challenge of a high speed and wide data bus, and have to overcome heavy signal loading to accommodate a reasonable number of ports for connecting multiple system devices.
2. *Cache coherence:* Cache memory is a critical component in SMP systems. In order to maintain data coherence among multiple caches, a cache coherence protocol must be incorporated into the bus protocol.
3. *Bus arbitration:* Each of the processors should be given an equal opportunity to gain access to the bus. Logic to prevent, detect, and resolve starvation and deadlock is typically needed.
4. *Bus bandwidth:* Besides faster and wider data buses, two other bandwidth-increasing features are important: pipelining bus requests using the split-transaction bus design, and transferring large data blocks in a *burst* transfer mode. As described before, the processing of a system bus request involves several distinct stages. Pipelining these requests can increase the effective bus bandwidth significantly.
5. *Memory access latency:* Memory access latency is a classic performance bottleneck. Techniques such as replicated arbiter and **bus parking** [Mahoney 1990] can reduce the latency in bus arbitration. Furthermore, early DRAM access before resolving cache coherence issues can reduce memory access latency. Such early DRAM accesses, however, may have to be canceled depending on the snoop results.
6. *Synchronization and locking:* SMP system buses need to provide an efficient way of implementing atomic read–modify–write instructions. Techniques such as address-based locking and nonblocking caches can reduce the interference between locking requests and other normal bus operations.
7. *Reliability and fault tolerance:* Parity or ECC can be used to detect and correct transmission errors. When an error is uncorrectable, the requester will receive a negative acknowledgment and will retry the request. The timeout scheme is commonly included to detect the loss of a command.

In the following, these important issues will be discussed in detail. Because the techniques for handling bus physics issues for the SMP system bus are basically the same as those in other buses, we will omit further discussion.

19.5.1 Cache Coherence Protocols

Cache memory is a critical component in SMP systems. It helps the processors to execute near their full speed by substantially reducing the average memory access time. This is achieved through fast cache hits for the majority of memory accesses and through reduced memory contention in the entire system. However, in designing a shared-memory SMP system where each processor is equipped with a cache memory, it is necessary to maintain coherence among the caches such that any memory access is guaranteed to return the latest version of the data in the system [Censier and Feautrier 1978]. Cache coherence can be enforced through a shared **snooping bus** [Goodman 1983, Sweazey and Smith 1986]. The basic idea is to rely on the broadcast nature of the bus to keep all the cache controllers informed of each other's activities so that they can perform the necessary operations to maintain coherency.

A number of snooping cache coherence protocols have been proposed [Archibald and Baer 1986, Sweazey and Smith 1986]. They can be broadly classified into the write-invalidate scheme and the write-broadcast scheme. In both schemes, read requests are carried out locally if a valid copy exists in the local cache. For write requests, these two schemes work differently. When a processor updates a cache line, all other copies of the same cache line must be invalidated according to the write-invalidate scheme to prevent other processors from accessing the stale data. Under the write-broadcast scheme, the new data of a write request will be broadcast to all the other caches to enable them to update any old copies. These two cache coherence schemes normally operate in conjunction with the *writeback* policy, because the *writethrough* policy generates memory traffic on every write request and is thus not suitable for a bus-based multiprocessor system.

The MESI (modified-exclusive-shared-invalid) write-invalidate cache coherence protocol with the writeback policy is considered in the following discussion. With minor variations, this protocol has been implemented in several commercial systems [Intel 1994, Greenley et al. 1995, Levitan et al. 1995]. In this protocol, each cache line has an associated MESI state recorded in the cache *directory* (also called cache tag array). The definitions of the four states are given in Table 19.1. When a memory request from either the processor or the snooping bus arrives at a cache controller, the cache directory is searched to determine cache hit/miss and the coherence action to be taken. The state transition diagram of the MESI protocol is illustrated in Figure 19.9, in which solid arrows represent state transitions due to requests issued by the local processor, and dashed arrows indicate state transitions due to requests from the snooping bus.

TABLE 19.1 Four States in MESI Coherence Protocol

State	Description
Modified (M)	The M-state indicates that the corresponding line is valid and is exclusive to the local cache. It also indicates that the line has been modified by the local processor. Therefore, the local cache has the latest copy of the line.
Exclusive (E)	The E-state indicates that the corresponding line is valid and is exclusive to the local cache. No modification has been made to the line. A write to an E-state line can be performed locally without producing any snooping bus traffic.
Shared (S)	The S-state indicates that the corresponding line is valid but may also exist in other caches in a multiprocessor system. Writing to an S-state line updates the local cache and generates a request to invalidate other shared copies.
Invalid (I)	The I-state indicates that the corresponding line is not available in the local cache. A cache miss occurs in accessing an I-state line. Typically, a line-fill request is issued to the memory to bring in the valid copy of the requested cache line.

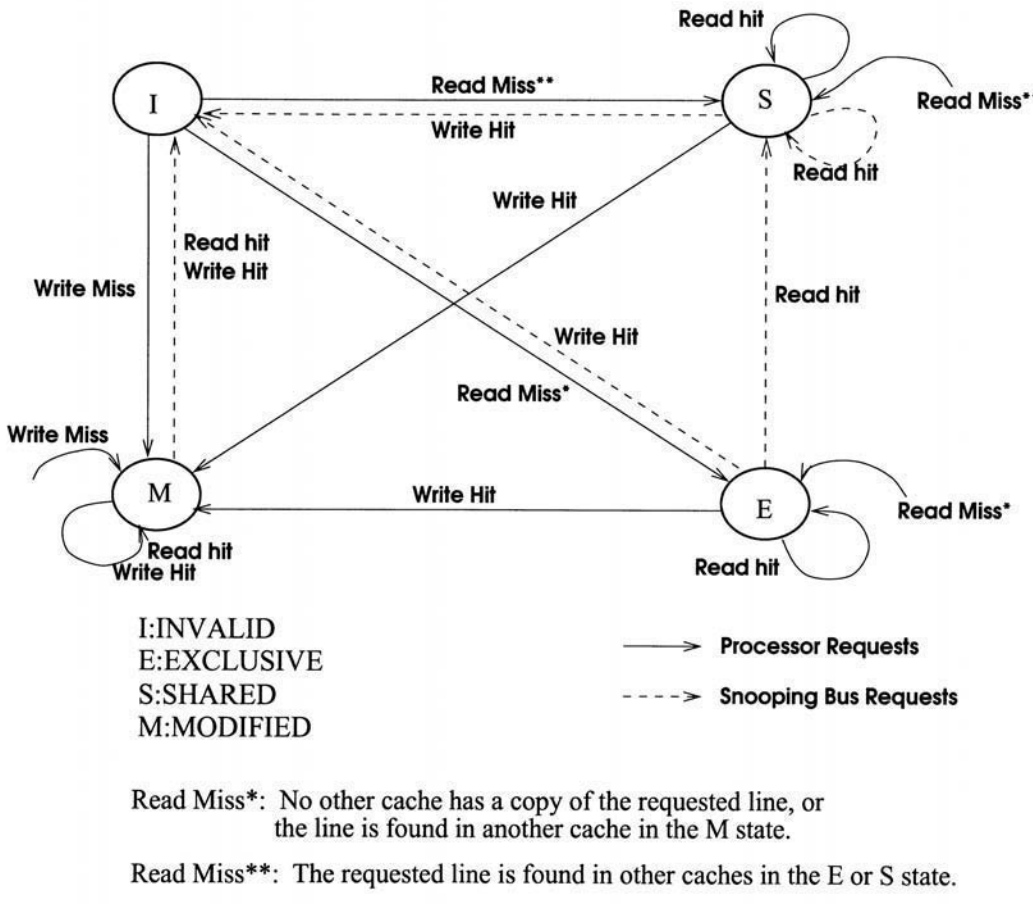

FIGURE 19.9 State transition diagram of MESI coherence protocol.

In general, when a read request from the processor hits a line in the local cache, the state of the cache line will not be altered. A write hit, on the other hand, will change the line state to M. In the meantime, if the original line state is S upon the write hit, an invalidation request will be issued to the snooping bus to invalidate the same line if it is present in any of the other caches. When a read miss occurs, the requested line will be brought into the cache in different states depending on whether the line is present in other caches and on its state in these caches. When a write miss occurs, the target line is fetched into the local cache; the new state becomes M and any copy of the cache line in any other cache is invalidated. For the requests from the snooping bus, a write hit always causes invalidation. A read hit to an M-state line will result in a writeback of the modified line and a transfer of ownership of the target line to the requesting processor. A read hit to an E- or S-state line for a snooping read request will cause a state transition to S.

19.5.2 Bus Arbitration

There are two issues in bus arbitration that are especially important for SMP system buses. The first is to ensure that the system is fair, deadlock-free, and starvation-free. In general, the fairness issue can be handled by using a first-come-first-served (FCFS) priority scheme, which will guarantee that requests are serviced in order of arrival. In fact, a simple random priority scheme may be adequate to provide a fair arbitration scheme for all the bus masters. The deadlock problem typically arises in a split-transaction bus where it is possible for a cycle of dependencies to form among the requests and replies. This can be handled

by distinguishing between requests and replies and always ensuring that replies are able to make forward progress. At first sight, it might seem that a fair arbitration policy should be starvation-free. However, this is not the case, because when overlapping bus requests are allowed, a request that has been granted by the bus may have to be later rejected due to interference with other requests. On a subsequent retry, the request may again encounter another conflict. In the pathological case, such a request may retry indefinitely. This situation is worse if there are lock requests and other resource conflicts on the system bus.

There are two general solutions to this starvation problem. The first solution eliminates any request rejection to prevent the starvation from happening. Whenever a conflict occurs, the request is queued at the location where the conflict is encountered and the queued request will be processed once the conflict condition disappears. This solution implies a variable-length bus pipeline, which requires extra handshaking on the system bus. In addition, excessive queues must be implemented to handle the conflict situation. The second solution depends on the ability to detect the starvation condition and to resolve the condition once it occurs. This method, in general, counts the number of times each request has been rejected. When a certain threshold is exceeded, emergency logic is activated to ensure that the starved request is serviced quickly and successfully.

The second important issue in bus arbitration is to minimize the latency for acquiring the bus. There are two techniques for doing this. The first technique is to replicate the arbiter in each processor. Each arbiter receives the request signals from all the bus masters just like in the centralized arbiter design. However, after the arbitration is resolved, each replicated arbiter only needs to send the grant signal to the local processor. The delay through a long wire across multiple chips in the centralized implementation can thus be avoided. The second latency-reduction technique is bus parking, which essentially implements a round-robin priority among the bus masters with a token being passed around. Once a bus master owns the token, it can access the bus without arbitration. This scheme is effective, in general, with a small number of bus masters.

19.5.3 Bus Bandwidth

An SMP system bus provides the only pathway for multiple processors to access the memory and the I/O devices. Therefore, the bandwidth of the system bus effectively determines the number of processors that can be supported. The simplest system bus design is to allow only one request at a time. A second command cannot be issued before the current bus request completes. This so-called *simple bus* design has very limited bus performance. A split-transaction bus, on the other hand, allows the overlapping of multiple bus requests using the pipelining technique. When a bus transaction is split, it only occupies the bus when the bus is really needed. For example, when a request is at the cache snooping stage or at the stage of accessing memory, the bus is available for other bus transactions. Even when the request is at the data transfer stage, the command/address bus is free to accept another request which does not require the data bus in the same cycle. The split-transaction approach utilizes the critical system bus much better. Most of the modern SMP system buses employ split-transaction designs to maximize the bus bandwidth [Peir et al. 1993, Galles and Williams 1994].

Even with the aggressive split-transaction design, the system bus can support a very limited number of today's high-performance microprocessors. For instance, consider a 50-MHz system bus connecting multiple 100 MIPS processors together. Assume that each processor has on-chip instruction/data caches as well as an external second-level cache; together, the instruction-per-miss rate is 50. In this case, each processor will generate bus traffic for handling two million cache misses every second. As a result, 25 processors will produce enough traffic to use up 100% of the bus bandwidth. This calculation does not allow for the fact that about 30% of the cache lines being replaced are typically modified and have to be written back to memory. In addition, this calculation does not take into account bus traffic due to I/O instructions, cache coherence transactions, etc. All these factors will further limit the number of processors that can be effectively supported by the system bus.

In the above discussion, two ideal conditions are assumed. The first assumption is that the bus utilization can reach 100%. In reality, the rule-of-thumb is that heavy queuing will occur when the bus utilization

reaches above 60%. The second assumption is that each bus request only occupies a single bus cycle. This is very difficult to achieve for a split-transaction bus running at 50 MHz. Typically, both the arbitration stage and the cache snooping stage may require more than one cycle. Even though these two stages do not occupy the bus directly, they need to access other related critical components such as the arbiter and the cache directory, respectively. Realistically, it takes a minimum of two cycles to initiate a new command on the system bus. This two-cycle-per-request design is already very aggressive, because the subsequent command is issued before the current command is acknowledged. In order to avoid potential interference between the active commands, protection hardware must be implemented. In addition, the data bus must be able to transfer a cache line in two cycles. For instance, a 128-bit data bus is required when the Intel Pentium processor is used in the above example, because the Pentium has 32-byte cache lines. Furthermore, any *idle* cycles during the data transfer must be eliminated. This typically requires a high-performance memory system with multiple modules each with independent memory banks.

Current projections suggest that processor performance will continue to improve at over 50% a year [Hennessy and Patterson 1996]. It is unlikely that improvements in the system bus will be able to keep up with the processor curve. Thus, the number of processors that the snooping bus can support is expected to decrease even further. There has been some work to extend the single bus architecture to multiple interleaved buses [Hopper et al. 1989] or hierarchical buses [Wilson 1987]. There have also been proposals to abandon the snooping bus approach and to use a directory method to enforce cache coherence [Lenoski et al. 1992, Gustavson 1992]. The detailed descriptions of these proposals are beyond the scope of this chapter.

19.5.4 Memory Access Latency

When a requested data item is not present in a processor's caches, the item must be fetched from memory. The delay in obtaining the data from memory may stall the requesting processor even when advanced techniques, such as out-of-order execution, nonblocking cache, relaxed consistency model, etc., are used to overlap processor execution with cache misses [Johnson 1990, Farkas and Jouppi 1992, Gharachorloo et al. 1990]. Therefore, it is important to design the system bus to minimize the number of cycles needed to return data upon a cache miss. Techniques such as replicated arbiter and bus parking to reduce arbitration cycles have been described in the subsection on Bus Arbitration above.

Another way to reduce the memory latency is to trigger DRAM access once the command arrives at the memory controller. This early DRAM access may have to be canceled if the requested line turns out to be present in a modified state in another processor's cache. Such a condition will only be known after the acknowledgment. Early DRAM access is very useful because the chance of hitting a modified line in another cache is not very high. In the unlikely case that the requested line is in fact modified in another cache, a cache-to-cache data and ownership transfer can be implemented to send the requested data directly to the requesting processor. The cache-to-cache transfer normally has higher priority and may bypass other requests in the write buffer of the processor that owns the modified line. In some implementations, the memory is also updated with the modified data during the cache-to-cache transfer.

19.5.5 Synchronization and Locking

The basic hardware primitive for implementing synchronization and locking is an atomic read–modify–write instruction such as Test&Set and Compare&Swap. These instructions must be guaranteed to read the contents of a memory location, test and modify the data, and write the result back to the same memory location in an indivisible sequence of operations. In a bus-based SMP system, the Test&Set instruction can be executed in two separate steps: a read-lock operation followed by a write-unlock operation. The read-lock operation reads the memory word and at the same time sets up certain hardware lock signals or registers so that no other requests for the same memory word will be permitted. After the memory word has been modified, the write-unlock operation returns the new result back to the memory location and releases the hardware lock.

There are two ways of implementing the hardware lock on the system bus. The first is to lock the bus completely during the period from the read-lock operation to the write-unlock operation; no other request is allowed in between. This approach has poor performance but may be suitable for a simple bus design which allows only one request at a time anyway. The second approach is to implement **address locking** on the system bus. When a read-lock operation is issued, an invalidation request is sent across the system bus to knock out any copy of the cache line from the other caches. In the meantime, the address of the cache line is recorded in a *lock register* to prevent any snooping on the same cache line until the lock is released by the write-unlock operation. Depending on the data alignment and the size of the synchronization variable, a read-lock operation may have to lock two cache lines at a time.

The address locking method minimizes the interference of a Test&Set instruction with other system bus requests, because only those requests that access the same cache line as the Test&Set will be rejected. Multiple lock requests, each from a different processor, are permitted as long as they target different cache lines. However, the lock request is still relatively expensive because the read-lock operation has to be broadcast to all the other processors and the issuing processor cannot proceed until confirmation is received from each processor.

19.6 Putting It All Together — CCL-XMP System Bus

CCL-XMP [Peir et al. 1993], an Intel Pentium-based SMP system, was designed and developed at the Computer and Communication Laboratories (CCL) of the Industrial Technology Research Institute (ITRI) of Taiwan under a collaborative effort with Acer, ICL, and Intel. The initial target system consists of eight 66-MHz Pentium processors, each with a 256-Kbyte second-level cache. The system bus, operating at 33 MHz, has independent control, command/address, and data buses. The MESI protocol is incorporated to enforce coherence among multiple caches. The system bus can arbitrate and accept one request every two cycles. A dual 64-bit data bus sustains a data transfer rate of over 400 Mbyte/s. Based on odd–even line interleaving, each data bus is connected to one memory module. In order to maximize the bus bandwidth, each memory module is divided into four independent banks. Furthermore, each bank is designed as a two-way interleaved DRAM array to provide zero-wait-state operation in the burst-mode data transfer at 33 MHz. CCL-XMP requires a powerful I/O subsystem. Both VESA local bus and PCI bus can be connected to the system bus through high-performance I/O bridges. The I/O bridges act as both a bus master and slave to transfer data between the system bus and the I/O buses.

The CCL-XMP system bus architecture is very similar to the general SMP bus architecture shown in Figure 19.8. Each request traverses through the same number of stages as described in Section 19.5. In this section, some of the relevant features of the CCL-XMP system bus design are described.

The CCL-XMP system bus is a synchronous bus. All the signals are driven at the rising edge of the clock and latched and sampled at the next rising edge. TTL voltage levels are used, and most signals are tristate with the exception of a few wire-OR control signals. The first version of the motherboard measures 32 cm $\times$ 32 cm. The system bus is 15 cm in length and supports eight bus slots. A single 64-bit data bus is included in the first prototype to support up to six Pentium processors. The clock tree is carefully laid out on the motherboard to limit the clock skew to within 2 ns. The total delay between the sender and the receiver through the system bus is less than 28 ns. This includes the delay of latches (flip-flops), the output TTL pad (with 120-pF bus loading), the input pad, boundary scan, clock skews, and other combinational logic. This is tolerable under the 33-MHz clock rate. In designing CCL-XMP, it is more challenging to manage the delay between the system bus interface chip and the second-level cache controller, which is operating at 66 MHz.

Figure 19.10 shows the detailed operation of a line-fill request on the CCL-XMP system bus. Arbitration of the command/address bus takes two cycles. Once the bus is granted, the memory read command and address are issued to the bus in the third cycle. It takes two cycles for all the bus masters (including I/O bridges) to search their cache directories and to update the respective line states upon a hit. At the same time, the memory controller initiates the DRAM access on the arrival of the read command. In the sixth cycle, an acknowledgment is sent back to the requester from each device based on the snooping result.

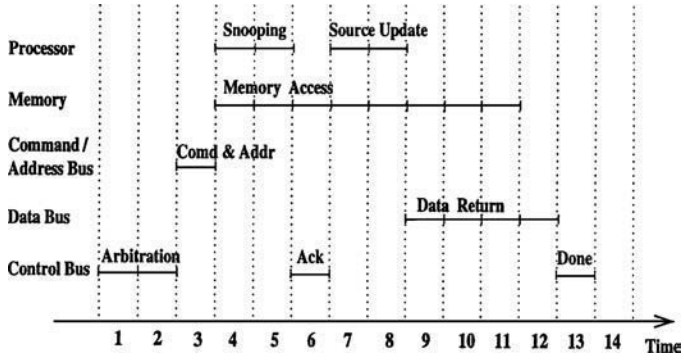

FIGURE 19.10 The pipeline cycles of a line-fill request on the CCL-XMP system bus.

The requester then updates its cache directory according to the acknowledgment and the cache coherence protocol. The memory controller begins arbitrating for the data bus two cycles before the data are fetched out of the DRAM array. A read request is given priority for using the data bus over a write request. After the data bus has been granted, the target 64-bit data is driven onto the bus in the ninth cycle. This is followed by three consecutive cycles to transfer the entire cache line to the requester. Finally, the command completes in the thirteenth cycle.

The CCL-XMP system bus uses replicated arbiters to achieve a two-cycle arbitration. Each arbiter receives *n* request signals, one from each bus master; but the grant signal is only sent to the local processor to avoid chip-crossing and long wiring delays. Basically, the arbiter latches the requests from all the bus masters at the end of the first cycle. In the second cycle, a resolution of the bus priority is carried out, and the winner is notified by the local arbiter. In addition, the CCL-XMP system bus uses the concept of *arbitration group* to achieve fairness without implementing FIFO queues. An arbitration group consists of the bus masters that are simultaneously requesting access to the system bus. The arbiter will serve all the members in the group according to some priority scheme. Any other master outside the current group is not allowed to join the group until all the requesters in the group have been granted the bus. After that, a new arbitration group can be formed.

Emergency logic is included in the arbiter design to resolve the starvation conditions. Each bus master has an associated *urgent counter*. The counter is incremented when a command receives a negative acknowledgment on the bus. A command can be rejected for several reasons, including resource busy, address locking, or some erroneous conditions. When the counter exceeds a certain threshold, the processor will raise its urgent request control line to the arbiter. The arbiter will then give the highest priority to the urgent requester and activate the emergency logic for resolving any blocking condition for the urgent request until the command is executed successfully and the urgent line is dropped.

The CCL-XMP system bus is a split-transaction bus; it allows the pipelining of multiple requests to sustain maximum bus bandwidth. Figure 19.11 illustrates two consecutive cache line-fill instructions on the command/address bus and the data transfer cycles for the two cache lines on the data buses. These two requests must have originated from different processors, because the Pentium processor only allows one outstanding memory request at a time. When the two requested lines are located in different memory modules, concurrent data transfers are possible. However, the data transfers have to be serialized when both accesses hit the same module. Note that there is a dead cycle on the data bus between the two cache line transfers on the same data bus. This is required to prevent a potential signal conflict. After the read command is issued, it takes six cycles including ECC check for the memory controller to return the target 64-bit data. Because the command bus can accept one command every two cycles, a dual 64-bit bus is designed to balance the performance. The dual data bus can transfer two 32-bytes of data every five bus cycles (including the dead cycle). This provides a sustained bus bandwidth of over 400 Mbyte/s.

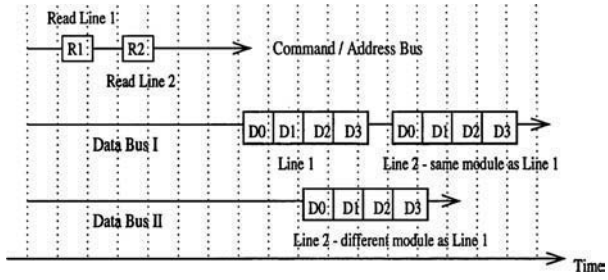

FIGURE 19.11 Split transactions on the CCL-XMP system bus.

CCL-XMP supports cache-to-cache data transfer with memory reflection when a request hits a modified line located in another cache. The memory copy is updated during the data transfer to the requester. The cache-to-cache transfer request has higher priority and may bypass other requests in the write buffer or the replacement writeback queues. This design reduces the penalty in accessing a modified line located in another processor's cache. The update of the memory copy provides the flexibility to transfer the ownership of the requested line to the requester and to switch the requested line to a shared state when there are subsequent read requests for the same line from other processors.

A pair of lock registers are implemented in each cache snooping controller to support address locking. At any given time, each processor can lock two cache lines to implement atomic read–modify–write memory instructions. A request is negative-acknowledged when a snoop hits an address in the lock register. The rejected request will be retried at a later time. Several protection registers are incorporated into the cache snooping controller to protect a cache line in a transient state from being accessed. For example, in addition to the lock registers, a current register (CR) is used to protect the active command until it completes. Also, a writeback register (WBR) records the modified line that is in the process of being written back to memory. As in the case of the lock register, a request will be rejected if it hits these protection registers. Parity bits are added to both the command/address bus and the data bus for detecting transmission errors.

19.7 Historical Perspective and Research Issues

Computer systems based on a shared backplane bus are very popular because they are both cost-effective and highly flexible. The early to mid 1980s saw the establishment of a number of standard buses such as the VME bus [Pri-Tal 1986], Fastbus [Gustavson 1986], Nubus [Taylor 1989], Multibus II [Mahoney 1990], and Futurebus [Borrill 1986]. All these bus architectures have been standardized by the IEEE to allow different vendors to design various boards that can be attached to each bus. Although the option of connecting multiple processors was considered in all these bus designs, only the Futurebus fully supports cache coherence protocols for both writethrough and writeback caches. Other important aspects, such as synchronous/asynchronous communication protocols, bus arbitration, bus bandwidth, locking mechanisms, TTL/BTL/ECL bus interfaces, connectors, pins, etc., are also significantly different among these open bus architectures. A comprehensive comparison can be found in [Borrill 1985].

The continuing search for higher-performance system interconnection has led to three more recent IEEE standards: the IEEE 896.x Futurebus+ [Aichinger 1992], the P1394 High-Performance Serial Bus [Teener 1992], and the P1596 Scalable Coherence Interface (SCI) [Gustavson 1992]. Futurebus+ is an expanded version of the original Futurebus (IEEE 896.1-1987). The work was initiated by the navy in 1988 and later gained widespread support from the working groups of the VMEbus and Multibus as well as many of the major computer companies. The Futurebus+ was designed to be a truly open standard. The standard defines in detail the architectural, electrical, and mechanical specifications. The data path of Futurebus+ varies from 32 to 256 bits wide to provide better performance scalability. The centralized arbitration mechanism is used for its simplicity and high performance. **Live insertion** is adopted to answer the needs of the market

for high-availability and fault-tolerant computers. In addition, Futurebus+ can be reconfigured as an I/O bus. Using the standard "Profile B" bridge, a Futurebus+ can be connected to other open-system buses.

Parallel backplane buses need large physical connectors. Such connectors are usually costly and are the primary source of failure. In addition, a point-to-point, unidirectional link can provide much faster transmission speed. The serial buses were introduced based on these advantages. The P1394 serial bus was first presented at IEEE CompCon 1992. It was designed for low cost, yet it provides the data transmission speed and low latency needed for a peripheral bus or as a possible backup of the backplane bus. The transmission protocol has three layers: transaction, link, and physical. Read, write, and lock are the three transactions supported, and each transaction can be divided into four stages: request, indication, response, and confirmation. The link layer provides a half-duplex data-packet delivery service. Each packet delivery needs to go through three stages: an arbitration stage to gain the access of the physical bus, a packet transmission stage to deliver the packet to the physical layer, and an acknowledgment stage to confirm the transmission with the receiver. The last stage is only required for synchronous package delivery. The split-transaction protocol and a Test&Set instruction are implemented in the P1394 serial bus to enable higher transaction rates and to properly handle locking activities.

As microprocessor performance increased at the rate of over 50% per year [Hennessy and Patterson 1996], it soon became apparent that the use of shared buses as the fundamental interconnection in multiprocessor systems creates a performance bottleneck. In July 1988, the P1596 SCI working group was formed with the aim of defining a scalable interconnect architecture for future shared-memory cache-coherent multiprocessor systems. There are a number of fundamental issues that must be solved in order to achieve this goal. First, signal speed has to be made independent of the size of the system. Second, multiple signal paths (links) have to be used so that multiple independent transfers can take place concurrently. Third, multiple cache coherence activities must be allowed to occur in parallel to overcome the bottleneck in the snooping-bus approach. To resolve the signal-speed problem, SCI uses point-to-point, unidirectional buses with low-voltage differential signals. *Ring* is the most common structure to connect multiple processing nodes (processors) through the fast SCI bus. *Distributed directory* is chosen to maintain cache coherence. The main memory and the directory which records the status of the associated lines in cache are distributed among all the nodes. Instead of maintaining full presence bits to indicate the caches where each line is located, a double link list is constructed to link all the copies of each cache line. Certain cache coherence actions, e.g., invalidation of shared copies of a cache line, need to go through the link list in a sequential fashion.

While all these standardization efforts were under way, a different approach was being taken by Rambus Inc. Rambus Inc. was set up in 1990 to develop technology that will enable systems to keep up with the increasing bandwidth requirements of processors [Rambus 1992]. The core of the Rambus technology is a proprietary chip-to-chip bus, the Rambus Channel. The Rambus Channel consists of a small number of very high-speed lines clocked at an aggressive 250 MHz. Data are transferred on both edges of the clock, allowing nine data lines to achieve a peak transfer rate of 500 Mbyte/s. A block-oriented protocol is used to allow effective utilization of this peak bandwidth. The key to achieving the high data rate lies in paying special attention to the physics of the bus. First, the Rambus interface consists of only 32 carefully terminated transmission lines. The small number of lines helps to reduce bus noise and power consumption, which is significant given the high clock rate. Second, clock-to-data skew is minimized by having a clock-to-master signal and a clock-from-master signal. The appropriate clock is chosen depending on the direction of the data transfer. Third, the channel is designed to operate with low-voltage swings, which further reduces power consumption. Finally, the maximum bus length is limited to about 10 cm and vertical surface-mount packages are used to enable devices to be densely packed onto the bus.

Although flexible and cost-effective, the bus is fundamentally not scalable in performance. This becomes increasingly apparent as processors become faster and SMP designs start pushing the performance limits of buses with fewer and fewer processors. Widening the data bus will not help, because of the bottleneck in the command/address bus. Using multiple buses [Hopper et al. 1989] to achieve more than one transfer at a time results in a complex bus interface design to maintain cache coherence with bus snooping mechanisms. The Scalable Coherence Interface uses a distributed-directory approach to build coherent shared-memory

multiprocessors with high-performance SCI rings. Although the SCI approach allows concurrent coherence activities, the cost of the large directory and the latency of coherence transaction remain serious problems. The Stanford DASH project [Lenoski et al. 1992] advocates the same distributed directory method using mesh-connected networks. Again, the latency in accessing remote memory may cause severe performance degradation [Kuskin et al. 1994]. The "right" way to build a scalable cache-coherent shared-memory multiprocessor remains an active research area.

An issue intimately related to bus performance is the performance of the memory subsystem. The standard RAS/CAS DRAM interface is designed for low cost and high density. In order to achieve sufficient memory bandwidth for today's demanding processors, an interleaved memory subsystem is often required. For SMPs, the memory interleaving is typically very aggressive. Recently, a couple of improvements to the existing DRAM interface have been announced. These include the extended-data-out (EDO) mode and the pipelined burst mode (PBM) [Kumanoya et al. 1995]. In addition, several new DRAM interfaces promising higher bandwidth and lower latency have been proposed. These include the synchronous DRAM, cached DRAM, Rambus DRAM, and others designed specifically for graphics application. The interested reader is referred to [Przybylski 1993, Kumanoya et al. 1995] for more information on these novel DRAM interfaces.

Defining Terms

Address locking: A mechanism to protect a specific memory address so that it can be accessed exclusively by a single processor.

Bus arbitration: The process of determining which competing bus master should be granted control of the bus.

Bus master: A bus device that is capable of initiating and controlling a communication on the bus.

Bus parking: A priority scheme which allows a bus master to gain control of the bus without arbitration.

Bus protocol: The set of rules which define precisely the bus signals that have to be asserted by the master and slave devices in each phase of a bus operation.

Bus slave: A bus device that can only act as a receiver.

Cache coherence protocol: A mechanism to maintain data coherence among multiple caches so that every data access will always return the latest version of that datum in the system.

Cache line: A block of data associated with a cache tag.

Live insertion: The process of inserting devices into a system or removing them from the system while the system is up and running.

Snooping bus: A multiprocessor bus that is continually monitored by the cache controllers to maintain cache coherence.

Split-transaction bus: A bus that overlaps multiple bus transactions, in contrast to the *simple bus* that services one bus request at a time.

Symmetric multiprocessor (SMP): A multiprocessor system where all the processors, memories, and I/O devices are equally accessible without a master–slave relationship.

References

Aichinger, B. P. 1992. Futurebus+ as an I/O bus: profile B, pp. 300–307. In *Proc. 19th Int. Symp. on Computer Architecture*.

Archibald, J. and Baer, J. L. 1986. Cache-coherence protocols: evaluation using a multiprocessor simulation model. *ACM Trans. Comput. Systems* 4(4):273–298.

Agarwal, A., Simoni, R., Hennessy, J., and Horowitz, M. 1988. An evaluation of directory schemes for cache coherence, pp. 280–289. In *Proc. 15th Int. Symp. on Computer Architecture*.

Borrill, P. 1985. MicroStandards special feature: a comparison of 32-bit buses. *IEEE Micro* 5(6):71–79.

Borrill, P. 1986. Futurebus: the ultimate in advanced system buses, pp. 210–216. In *Proc. Buscon '86 West*.

Censier, L. and Feautrier, P. 1978. A new solution to coherence problems in multicache systems. *IEEE Trans. Comput*. C-27(12):1112–1118.

Chaiken, D., Fields, C., Kurihara, K., and Agarwal, A. 1990. Directory-based cache coherence in large-scale multiprocessors. *IEEE Comput*. 23(6):49–59.
Farkas, K. and Jouppi, N. 1992. Complexity/performance tradeoffs with non-blocking loads, pp. 211–222. In *Proc. 21st Int. Symp. on Computer Architecture*.
Galles, M. and Williams, E. 1994. Performance optimizations, implementation, and verification of the SGI challenge multiprocessor, pp. 134–143. In *Proc. 1994 Hawaii Int. Conf. on System Science, Architecture Track*.
Gharachorloo, K., et. al. 1990. Memory consistency and event ordering in scalable shared-memory multiprocessors, pp. 15–26. In *Proc. 17th Int. Symp. on Computer Architecture*.
Giacomo, J. D. 1990. *Digital Bus Handbook*. McGraw–Hill, New York.
Goodman, J. 1983. Using cache memory to reduce processor-memory traffic, pp. 124–131. In *Proc. 10th Int. Symp. on Computer Architecture*.
Greenley, D. et. al. 1995. Ultrasparc: the next generation superscalar 64-bit SPARC, pp. 442–451. In *Proc. COMPCON'95*.
Gustavson, D. B. 1984. Computer buses — a tutorial. *IEEE Micro* (4):7–22.
Gustavson, D. B. 1986. Introduction to the Fastbus. *Microprocessors and Microsystems* 10(2):77–85.
Gustavson, D. B. 1992. The scalable coherent interface and related standards projects. *IEEE Micro* 12(1): 10–22.
Gustavson, D. B. and Theus, J. 1983. Wire-OR logic on transmission lines. *IEEE Micro* 3(3):51–55.
Hennessy, J. and Patterson, D. 1996. *Computer Architecture, a Quantitative Approach*, 2nd ed. Morgan Kaufmann, San Francisco.
Hopper, A., Jones, A., and Lioupis, D. 1989. Multiple vs wide shared bus multiprocessors, pp. 300–306. In *Proc. 16th Int. Symp. on Computer Architecture*.
IBM Corp. 1982. IBM 3081 Functional Characteristics, GA22-7076. IBM Corp., Poughkeepsie, NY.
Intel Corp. 1994. *Pentium Processor User's Manual*, Vols. 1, 2. Intel Corp. Order nos. 241428, 241429.
Johnson, M. 1990. *Superscalar Microprocessor Design*. Prentice–Hall, Englewood Cliffs, NJ.
Kumanoya, M., Ogawa, T., and Inoue, K. 1995. Advances in DRAM interfaces. *IEEE Micro* 15(6):30–36.
Kuskin, J., et. al. The Stanford FLASH multiprocessor, pp. 302–313. In *Proc. 21st Int. Symp. on Computer Architecture*.
Lenoski, D. et. al. 1992. The Stanford Dash multiprocessor. *IEEE Comput*. 25(3):63–79.
Levitan, D., Thomas, T., and Tu, P. 1995. The PowerPC 620 microprocessor: a high performance superscalar RISC microprocessor, pp. 285–291. In *Proc. COMPCON '95*.
Mahoney, J. 1990. Overview of Multibus II architecture. *SuperMicro J*. No. 4, pp. 58–67.
Metcalfe, R. M., and Boggs, D. R. 1976. Ethernet: distributed packet switching for local computer networks. *Commun. ACM* 19(7):395–404.
Peir, J. K., et al. 1993. CCL-XMP: a Pentium-based symmetric multiprocessor system, pp. 545–550. In *Proc. 1993 Int. Conf. on Parallel and Distributed Systems*.
Pri-Tal, S. 1986. The VME subsystem bus. *IEEE Micro* 6(2):66–71.
Przybylski, S. 1993. DRAMs for new memory systems. parts 1, 2, 3. *Microprocessor Rep*., Mar. 8, pp. 18–21.
Rambus. 1992. *Rambus Architectural Overview*. Rambus, Inc., Mountain View, CA.
Stenstorm, P. 1990. A survey of cache coherence schemes for multiprocessors. *IEEE Comput*. 23(6):12–25.
Sweazey, P. and Smith, A. J. 1986. A class of compatible cache consistency protocols and their support by IEEE Futurebus, pp. 414–423. In *Proc. 13th Int. Symp. on Computer Architecture*.
Taub, D. M. 1983a. Overcoming the effects of spurious pulses on wired-OR lines in computer bus systems. *Electron. Lett.* 19(9):340–341.
Taub, D. M. 1983b. Limitations of looped-line scheme for overcoming wired-OR glitch effects. *Electron. Lett.* 19(15):579–580.
Taub, D. M. 1984. Arbitration and control acquisition in the proposed IEEE 896 FutureBus. *IEEE Micro* 4(4):28–41.
Taylor, B. G. 1989. Developing for the Macintosh NuBus, pp. 143–175. In *Proc. Eurobus/UK Conference*.

Teener, M. 1992. A bus on a diet — the serial bus alternative: an introduction to the P1394 high performance serial bus, pp. 316–321. In *Compcon '92*.
Wilson, A. 1987. Hierarchical cache/bus architecture for shared memory multiprocessors, pp. 244–252. In *Proc. 14th Int. Symp. on Computer Architecture*.
Zalewski, J. 1995. *Advanced Multimicroprocessor Bus Architecture*. IEEE Computer Society Press.

Further Information

Advanced Multimicroprocessor Bus Architectures by J. Zalewski [Zalewski 1995] contains a comprehensive collection of papers covering bus basics, physics, arbitration and protocols, board and interface designs, cache coherence, various standard bus architectures, and their performance evaluations. The bibliography section at the end provides a complete list of references for each of the topics discussed in the book.

Digital Bus Handbook by J. D. Giacomo [Giacomo 1990] is a good source of information for the details of the various standard bus architectures. In addition, this book has several chapters devoted to the electrical and mechanical issues in bus design.

The bimonthly journal *IEEE Micro* and the *Proceedings of International Symposium on Computer Architecture* are good sources of the latest papers in computer architecture area including computer buses.

20

Input/Output Devices and Interaction Techniques

20.1 Introduction
20.2 Interaction Tasks, Techniques, and Devices
20.3 The Composition of Interaction Tasks
20.4 Properties of Input Devices
20.5 Discussion of Common Pointing Devices
20.6 Feedback and Perception — Action Coupling
20.7 Keyboards, Text Entry, and Command Input
20.8 Modalities of Interaction
Voice and Speech • Pen-Based Gestures and Hand Gesture Input • Bimanual Input • Passive Measurement: Interaction in the Background
20.9 Displays and Perception
Properties of Displays and Human Visual Perception
20.10 Color Vision and Color Displays
20.11 Luminance, Color Specification, and Color Gamut
20.12 Information Visualization
General Issues in Information Coding • Color Information Coding • Integrated Control/Display Objects • Three-Dimensional Graphics and Virtual Environments • Augmented Reality
20.13 Scale in Displays
Small Displays • Multiple Displays • Large-Format Displays
20.14 Force and Tactile Displays
20.15 Auditory Displays
Nonspeech Audio • Speech Output • Spatialized Audio Displays
20.16 Future Directions

Ken Hinckley
Microsoft Research

Robert J. K. Jacob
Tufts University

Colin Ware
University of New Hampshire

20.1 Introduction

The computing literature often draws an artificial distinction between input and output; computer scientists are used to regarding a screen as a passive output device and a mouse as a pure input device. However, nearly all examples of human–computer interaction require *both* input and output to do anything useful.

For example, what good would a mouse be without the corresponding feedback embodied by the cursor on the screen, as well as the sound and feel of the buttons when they are clicked? The distinction between output devices and input devices becomes even more blurred in the real world. A sheet of paper can be used both to record ideas (input) and to display them (output). Clay reacts to the sculptor's fingers, yet it also provides feedback through the curvature and texture of its surface. Indeed, the complete and seamless integration of input and output is becoming a common research theme in advanced computer interfaces, such as ubiquitous computing [Weiser, 1991] and tangible interaction [Ishii and Ullmer, 1997].

Input and output bridge the chasm between a computer's inner world of bits and the real world perceptible to the human senses. *Input* to computers consists of sensed information about the physical environment. Familiar examples include the mouse, which senses movement across a planar surface, and the keyboard, which detects a contact closure when the user presses a key. However, any sensed information about physical properties of people, places, or things can serve as input to computer systems. *Output* from computers can comprise any emission or modification to the physical environment, such as a display (including the cathode ray tube [CRT], flat-panel displays, or even light-emitting diodes), speakers, or tactile and force feedback devices (sometimes referred to as *haptic* displays). An *interaction technique* is the fusion of input and output, consisting of all hardware and software elements, that provides a way for the user to accomplish a task. For example, in the traditional graphical user interface (GUI), users can scroll through a document by clicking or dragging the mouse (input) within a scroll bar displayed on the screen (output).

The fundamental task of human–computer interaction is to shuttle information between the brain of the user and the silicon world of the computer. Progress in this area attempts to increase the useful bandwidth across that interface by seeking faster, more natural, and more convenient means for users to transmit information to computers, as well as efficient, salient, and pleasant mechanisms to provide feedback to the user. On the user's side of the communication channel, interaction is constrained by the nature of human attention, cognition, and perceptual–motor skills and abilities; on the computer side, it is constrained only by the technologies and methods that we can invent.

Research in input and output focuses on the two ends of this channel: the devices and techniques computers can use for communicating with people, and the perceptual abilities, processes, and organs people can use for communicating with computers. It then attempts to find the common ground through which the two can be related by studying new modes of communication that could be used for human–computer interaction (HCI) and developing devices and techniques to use such modes. Basic research seeks theories and principles that inform us of the parameters of human cognitive and perceptual facilities, as well as models that can predict or interpret user performance in computing tasks. Advances can be driven by the need for new modalities to support the unique requirements of specific applications, by technological breakthroughs that HCI researchers attempt to apply to improving or extending the capabilities of interfaces, by theoretical insights suggested by studies of human abilities and behaviors, or even by problems uncovered during careful analyses of existing interfaces. These approaches complement one another; all have their value and contributions to the field, but the best research seems to have elements of all of these.

20.2 Interaction Tasks, Techniques, and Devices

A designer looks at the interaction tasks necessary for a particular application [Foley et al., 1984]. Interaction tasks are low-level primitive inputs required from the user, such as entering a text string or choosing a command. For each such task, the designer chooses an appropriate interaction technique. In selecting an interaction device and technique for each task in a human–computer interface, simply making an optimal choice for each task individually may lead to a poor overall design, with too many different or inconsistent types of devices or dialogues. Therefore, it is often desirable to compromise on individual choices to reach a better overall design.

There may be several different ways to accomplish the same task. For example, one could use a mouse to select a command by using a pop-up menu, a fixed menu (a palette or command bar), multiple clicking,

circling the desired command, or even writing the name of the command with the mouse. Software might detect patterns of mouse use in the background, such as repeated surfing through menus, and automatically suggest commands or help topics [Horvitz et al., 1998]. The latter suggests a shift from the classical view of interaction as direct manipulation, in which the user is responsible for all actions and decisions, to one that uses background sensing techniques to allow technology to support the user with semiautomatic or implicit actions and services [Buxton, 1995a].

20.3 The Composition of Interaction Tasks

Early efforts in human–computer interaction sought to identify elemental tasks that appear repeatedly in human–computer dialogs. Foley et al. [1984] proposed that user interface transactions are composed of the following elemental tasks:

Selection — Choosing objects from a set of alternatives
Position — Specifying a position within a range, including picking a screen coordinate with a pointing device
Orientation — Specifying an angle or three-dimensional orientation
Path — Specifying a series of positions or orientations over time
Quantification — Specifying an exact numeric value
Text — Entering symbolic data

While these are commonly occurring tasks in many direct-manipulation interfaces, a problem with this approach is that the level of analysis at which one specifies elemental tasks is not well defined. For example, for position tasks, a screen coordinate could be selected using a pointing device such as a mouse, but it might be entered as a pair of numeric values (quantification) using a pair of knobs (like an Etch-a-Sketch) where precision is paramount. But if these represent elemental tasks, why do we find that we must subdivide position into a pair of quantification subtasks for some devices but not for others?

Treating all tasks as hierarchies of subtasks, known as *compound tasks*, is one way to address this. With appropriate design, and by using technologies and interaction metaphors that parallel as closely as possible the way the user thinks about a task, the designer can phrase together a series of elemental tasks into a single cognitive chunk. For example, if the user's task is to draw a rectangle, a device such as an Etch-a-Sketch is easier to use. For drawing a circle, a pen is far easier to use. Hence, the choice of device influences the level at which the user is required to think about the individual actions that must be performed to achieve a goal. See Buxton [1986] for further discussion of this important concept.

A problem with viewing tasks as assemblies of elemental tasks is that typically this view only considers explicit input in the classical direct-manipulation paradigm. Where do devices like cameras, microphones, and fingerprint scanners fit in? These support higher-level data types and concepts (e.g., images, audio, and identity). Advances in technology will continue to yield new "elemental" inputs. However, these new technologies also may make increasing demands on systems to move from individual samples to synthesis of meaningful structure from the resulting data [Fitzmaurice et al., 1999].

20.4 Properties of Input Devices

The breadth of input devices and displays on the market today can be completely bewildering. Fortunately, there are a number of organizing properties and principles that can help to make sense of the design space and performance issues. First, we consider continuous, manually operated pointing devices (as opposed to discrete input mechanisms such as buttons or keyboards, or other devices not operated with the hand, which we will discuss briefly later). For further insight, readers may also wish to consult complete taxonomies of devices [Buxton, 1983; Card et al., 1991]. As we shall see, however, it is nearly impossible to describe properties of input devices without reference to output — especially the resulting feedback on

the screen — because, after all, input devices are only useful insofar as they support interaction techniques that allow the user to accomplish something.

Physical property sensed — Traditional pointing devices typically sense position, motion, or force. A tablet senses position, a mouse measures motion (i.e., change in position), and an isometric joystick senses force. An isometric joystick is a self-centering, force-sensing joystick such as the IBM TrackPoint ("eraser-head") found on many laptops. For a rotary device, the corresponding properties are angle, change in angle, and torque. Position-sensing devices are also known as *absolute* devices, whereas motion-sensing devices are *relative* devices. An absolute device can fully support relative motion, because it can calculate changes to position, but a relative device cannot fully support absolute positioning. In fact, it can only emulate position at all by introducing a cursor on the screen. Note that it is difficult to move the mouse cursor to a particular area of the screen (other than the edges) without looking at it, but with a tablet one can easily point to a region with the stylus using the kinesthetic sense [Balakrishnan and Hinckley, 1999]. This phenomenon is known informally as *muscle memory*.

Transfer function — In combination with the host operating system, a device typically modifies its signals using a mathematical transformation that scales the data to provide smooth, efficient, and intuitive operation. An *appropriate mapping* is a transfer function that matches the physical properties sensed by the input device. Appropriate mappings include force-to-velocity, position-to-position, and velocity-to-velocity functions. For example, an isometric joystick senses force; a nonlinear rate mapping transforms this into a velocity of movement [Rutledge and Selker, 1990; Zhai and Milgram, 1993; Zhai et al., 1997]. When using a rate mapping, the device ideally should also be *self-centering* (i.e., spring return to the zero input value), so that the user can stop quickly by releasing the device. A common inappropriate mapping is calculating a speed of scrolling based on the position of the mouse cursor, such as extending a selected region by dragging the mouse close to the edge of the screen. The user has no feedback about when or to what extent scrolling will accelerate, and the resulting interaction can be hard to learn how to use and difficult to control.

Gain — This is a simple multiplicative transfer function, which can also be described as a control–display (C:D) ratio: the ratio between the movement of the input device and the corresponding movement of the object it controls. For example, if a mouse (the control) must be moved 1 cm on the desk in order to move a cursor 2 cm on the screen (the display), the device has a 1:2 control–display ratio. However, on commercial pointing devices and operating systems, the gain is rarely constant*; an *acceleration function* is often used to modulate the gain depending on velocity. Acceleration function is simply another term for a transfer function that exhibits an exponential relationship between velocity and gain. Experts believe the primary benefit of acceleration is to reduce the footprint, or the physical movement space, required by an input device [Jellinek and Card, 1990; Hinckley et al., 2001]. One must also be very careful when studying the possible influence of gain settings on user performance: experts have criticized gain as a fundamental concept, since it confounds two separate concepts (device size and display size) in one arbitrary metric [Accot and Zhai, 2001]. Furthermore, user performance may exhibit speed–accuracy trade-offs, calling into question the assumption that there exists an optimal C:D ratio [MacKenzie, 1995].

Number of dimensions — Devices can measure one or more linear and angular dimensions. For example, a mouse measures two linear dimensions, a knob measures one angular dimension, and a six-degree-of-freedom magnetic tracker measures three linear dimensions and three angular. If the number of dimensions required by the user's interaction task does not match the number of dimensions provided by the input device, then special handling (e.g., interaction techniques that may require extra buttons, graphical widgets, mode switching, etc.) must be introduced.

*Direct input devices are an exception, since the C:D ratio is typically fixed at 1:1, but see also Sears and Shneiderman [1991].

This is a particular concern for three-dimensional user interfaces and interaction [Zhai, 1998; Hinckley et al., 1994b]. Numerous interaction techniques have been proposed to allow standard 2-D pointing devices to control 3-D positioning or orientation tasks (e.g., Conner et al., 1992; Chen et al., 1988, Bukowski and Sequin, 1995). Well designed interaction techniques using specialized multiple degree-of-freedom input devices can sometimes offer superior performance [Ware and Rose, 1999; Hinckley et al., 1997] but may be ineffective for standard desktop tasks, so overall performance must be considered [Balakrishnan et al., 1997; Hinckley et al., 1999].

Pointing speed and accuracy — The standard way to characterize pointing device performance employs the Fitts' law paradigm [Douglas et al., 1999]. Fitts' law relates the movement time to point at a target, the amplitude of the movement (the distance to the target), and the width of the target (i.e., the precision requirement of the pointing movement). While not emphasized in this chapter, Fitts' law is the single most important quantitative analysis, testing, and prediction tool available to input research and device evaluation. For an excellent overview of its application to the problems of HCI, including use of Fitts' law to characterize bandwidth (a composite measure of both speed and accuracy), see MacKenzie [1992]. For discussion of other accuracy metrics, see MacKenzie et al. [2001].

Recent years have seen a number of new insights and new applications for Fitts' law. Fitts' law was originally conceived in the context of rapid, aimed movements, but it can also be applied to tasks such as scrolling [Hinckley et al., 2001], multiscale navigation [Guiard et al., 2001], and crossing boundaries [Accot and Zhai, 2002]. Recently, researchers have also applied Fitts' law to expanding targets that double in width as the user approaches them. Even if the expansion begins after the user has already covered 90% of the distance from a starting point, the expanding target can be selected as easily as if it had been fully expanded since the movement began [McGuffin and Balakrishnan, 2002]. However, it remains unclear whether this can be successfully applied to improving pointing performance for multiple targets that are closely packed together, as typically found in menus and tool palettes. For tasks that exhibit continuous speed–accuracy requirements, such as moving through a hierarchical menu, Fitts' law cannot be applied, but researchers have recently formulated the *steering law,* which does addresses such tasks [Accot and Zhai, 1997; Accot and Zhai, 1999; Accot and Zhai 2001].

Input device states — To select a single point or region with an input device, users need a way to signal when they are selecting something, as opposed to when they are just moving over something to reach a desired target. The need for this fundamental signal of intention is often forgotten by researchers eager to explore new interaction modalities such as empty-handed pointing (e.g., using camera tracking or noncontact proximity sensing of hand position). The *three-state model* of input generalizes the states sensed by input devices [Buxton, 1990b] as *tracking*, which causes the cursor to move; *dragging*, which allows the selection of objects by clicking or the moving of objects by clicking and dragging them; and *out of range*, which occurs when the device moves out of its physical tracking range (e.g., a mouse is lifted from the desk, or a stylus is removed from a tablet). Most pointing devices sense only two of these three states: for example, a mouse senses tracking and dragging, but a touchpad senses tracking and the out-of-range state. Hence, to fully simulate the functionality offered by mice, touchpads need special procedures, such as tapping to click, which are prone to inadvertent activation (e.g., touching the pad by accident causes a click). For further discussion and examples, see Buxton, 1990b; Hinckley et al., 1998a; and MacKenzie and Oniszczak, 1998.

Direct vs. indirect control — A mouse is indirect (you move it on the table to point to a spot on the screen); a touchscreen is direct (the display surface is also the input surface). Direct devices raise several unique issues. Designers must consider the possibility of parallax error resulting from a gap between the input and display surfaces, reduced transmissivity of the screen introduced by a sensing layer, or occlusion of the display by the user's hands. Another issue is that touchscreens can support a cursor tracking state or a dragging state, but not both. Typically, touchscreens move directly from the out-of-range state to the dragging state when the user touches the screen, with

no intermediate cursor feedback [Buxton, 1990b]. Techniques for touchscreen cursor feedback typically require that selection occurs on lift-off [Sears et al., 1991; Sears et al., 1992]. See also "Pen input" in Section 20.5 of this chapter.

Hardware criteria — Various other characteristics can distinguish input devices but are perhaps less important in distinguishing the fundamental types of interaction techniques that can be supported. Engineering parameters of a device's performance, such as sampling rate, resolution, accuracy, and linearity, can all influence performance, as well. *Latency* is the end-to-end delay between the user's physical movement, sensing this, and providing the ultimate system feedback to the user. Latency can be a devious problem because it is impossible to eliminate it completely from system performance. Latency of more than 75 to 100 milliseconds significantly impairs user performance for many interactive tasks [Robertson et al., 1989; MacKenzie and Ware, 1993]. For vibrotactile or haptic feedback, users may be sensitive to much smaller latencies of just a few milliseconds [Cholewiak and Collins, 1991].

Other user performance criteria — Devices can also be distinguished using various criteria: user performance, learning time, user preference, and so forth. Device *acquisition time*, which is the average time to pick up or put down a device, is often assumed to be a significant factor in user performance, but the Fitts' law bandwidth of a device tends to dominate this unless switching occurs frequently [Douglas and Mithal, 1994]. However, one exception is stylus- or pen-based input devices; pens are generally comparable to mice in general pointing performance [Accot and Zhai 1999] or even superior for some high-precision tasks [Guiard et al., 2001], but these benefits can easily be negated by the much greater time it takes to switch between using a pen and using a keyboard.

20.5 Discussion of Common Pointing Devices

Here, we briefly describe commonly available pointing devices and some issues that can arise with them in light of the properties discussed above.

Mouse — A mouse senses movement relative to a flat surface. Mice exhibit several properties that are well suited to the demands of desktop graphical interfaces [Balakrishnan et al., 1997]. The mouse is stable and does not fall over when released (unlike a stylus on a tablet). A mouse can also provide integrated buttons for selection, and because the force required to activate a mouse's buttons is orthogonal to the plane of movement, it helps to minimize accidental clicking or interference with motion. Another subtle benefit is the possibility for users to employ a combination of finger, hand, wrist, arm, and even shoulder muscles to span the range of tasks from short precise selections to large, ballistic movements [Zhai et al., 1996; Balakrishnan and MacKenzie, 1997]. Finally, Fitts' law studies show that users can point with the mouse about as well as with the hand itself [Card et al., 1978].

Trackball — A trackball is like a mouse that has been turned upside down, with a mechanical ball that rolls in place. The main advantage of trackballs is that they can be used on an inclined surface, and they often require a smaller footprint than mice. They also employ different muscle groups, which some users find more comfortable. However, trackballs cannot accommodate buttons as easily as mice, so tasks that require moving the trackball while holding down a button can be awkward [MacKenzie et al., 1991].

Tablets — Most tablets sense the absolute position of a mechanical intermediary such as a stylus or puck on the tablet surface. A *puck* is a mouse that is used on a tablet; the only difference is that it senses absolute position and it cannot be used on a surface other than the tablet. Absolute mode is generally preferable for tasks such as tracing, digitizing, drawing, free-hand inking, and signature capture. Tablets that sense contact with the bare finger are known as *touch tablets* [Buxton et al., 1985]. Touchpads are miniature touch tablets, as commonly found on portable computers [MacKenzie and Oniszczak 1998]. A touchscreen is a transparent, touch-sensitive tablet mounted on a display, but it demands different handling than a tablet (see "Direct vs. indirect control" in Section 20.4).

Pen input — Pen-based input for mobile devices is an area of increasing practical concern. Pens effectively support activities such as inking, marking, and gestural input (see Section 20.8.2), but pens raise a number of problems when supporting graphical interfaces originally designed for mouse input. Pen input raises the concerns about direct input devices described previously. There is no way to see exactly what position will be selected before selecting it: pen contact with the screen directly enters the dragging state of the three-state model [Buxton, 1990b]. There is neither a true equivalent of a hover state for tool tips nor an extra button for context menus. Pen dwell time on a target can be used to provide one of these two functions. For detecting double-tap, allow a longer interval between the taps (as compared to double-click on a mouse), and also allow a significant change to the screen position between taps. Finally, users often want to touch the screen of small devices using a bare finger, so applications should be designed to accommodate imprecise selections. Note that some pen-input devices, such as the Tablet PC, use an inductive sensing technology that can only sense contact from a specially instrumented stylus, and thus cannot be used as a touchscreen. However, this deficiency is made up for by the ability to track the pen when it is close to (but not touching) the screen, allowing support for a tracking state with cursor feedback (and hence tool tips).

Joysticks — There are many varieties of joystick. As mentioned earlier, an isometric joystick senses force and returns to center when released. Because isometric joysticks can have a tiny footprint, they are often used when space is at a premium, allowing integration with a keyboard and hence rapid switching between typing and pointing [Rutledge and Selker, 1990; Douglas and Mithal, 1994]. Isotonic joysticks sense the stick's angle of deflection, so they tend to move more than isometric joysticks, offering better feedback to the user. Such joysticks may or may not have a mechanical spring return to center. Some joysticks include force sensing, position sensing, and other special features. For a helpful organization of the complex design space of joysticks, see Lipscomb and Pique [1993].

Alternative devices — Researchers have explored using the feet [Pearson and Weiser, 1988], head tracking, and eye tracking as alternative approaches to pointing. Head tracking has much lower pointing bandwidth than the hands and may require the neck to be held in an awkward fixed position, but it has useful applications for intuitive coupling of head movements to virtual environments [Sutherland, 1968; Brooks, 1988] and interactive 3-D graphics [Hix et al., 1995; Ware et al., 1993]. Eye movement–based input, properly used, can provide an unusually fast and natural means of communication, because we move our eyes rapidly and almost unconsciously. The human eye fixates visual targets within the fovea, which fundamentally limits the accuracy of eye gaze tracking to 1 degree of the field of view [Zhai et al., 1999]. Eye movements are subconscious and must be interpreted carefully to avoid annoying the user with unwanted responses to his actions, known as the *Midas touch problem* [Jacob, 1991]. Eye-tracking technology is expensive and has numerous technical limitations, confining its use to research labs and disabled persons with few other options.

20.6 Feedback and Perception — Action Coupling

The ecological approach to human perception [Gibson, 1986] asserts that the organism, the environment, and the tasks the organism performs are inseparable and should not be studied in isolation. Hence, perception and action are intimately linked in a single motor–visual feedback loop, and any separation of the two is artificial. The lesson for interaction design is that techniques must consider both the motor control (input) and feedback (output) aspects of the design and how they interact with one another.

From the technology perspective, one can consider feedback passive or active. Active feedback is under computer control. This can be as simple as presenting a window on a display, or as sophisticated as simulating haptic contact forces with virtual objects when the user moves an input device. We will discuss active feedback techniques later in this chapter, when we discuss display technologies and techniques.

Passive feedback may come from sensations within the user's body, as influenced by physical properties of the device, such as the shape, color, and feel of buttons when they are depressed. The industrial design of a device suggests the purpose and use of a device even before a user touches it [Norman, 1990]. Mechanical sounds and vibrations that result from using the device provide confirming feedback of the user's action. The shape of the device and the presence of landmarks can help users orient a device without having to look at it [Hinckley et al., 1998b]. Proprioceptive and kinesthetic feedback are somewhat imprecise terms, often used interchangeably, that refer to sensations of body posture, motion, and muscle tension [Burdea, 1996]. These senses allow users to feel how they are moving an input device without looking at the device, and indeed without looking at the screen in some situations [Mine et al., 1997; Balakrishnan and Hinckley, 1999]. This may be important when the user's attention is divided between multiple tasks and devices [Fitzmaurice and Buxton, 1997]. Sellen et al. [1992] report that muscular tension from depressing a foot pedal makes modes more salient to the user than purely visual feedback. Although all of these sensations are passive and not under the direct control of the computer, these examples nonetheless demonstrate that they are relevant to the design of devices. Interaction techniques can consider these qualities and attempt to leverage them.

User performance may be influenced by correspondences between input and output. Some correspondences are obvious, such as the need to present confirming visual feedback in response to the user's actions. Ideally, feedback should indicate the results of an operation before the user commits to it (e.g., highlighting a button or menu item when the cursor moves over it). Kinesthetic correspondence and perceptual structure, described below, are less obvious.

Kinesthetic correspondence refers to the principle that graphical feedback on the screen should correspond to the direction in which the user moves the input device, particularly when 3-D rotation is involved [Britton et al., 1978]. Users can easily adapt to certain noncorrespondences: when the user moves a mouse forward and back, the cursor actually moves up and down on the screen; if the user drags a scrollbar downward, the text on the screen scrolls upward. With long periods of practice, users can adapt to almost anything. For example, for more than 100 years psychologists have known of the phenomenon of prism adaptation: people can eventually adapt to wearing prisms that cause everything to look upside down [Stratton, 1897]. However, one should not force users to adapt to a poor design.

Researchers also have found that the interaction of the input dimensions of a device with the control dimensions of a task can exhibit perceptual structure. Jacob et al. [1994] explored two input devices: a 3-D position tracker with integral (x, y, z) input dimensions and a standard 2-D mouse, with (x, y) input separated from (z) input by holding down a mouse button. For selecting the position and size of a rectangle, the position tracker is most effective. For selecting the position and grayscale color of a rectangle, the mouse is most effective. The best performance results when the integrality or separability of the input matches that of the output.

20.7 Keyboards, Text Entry, and Command Input

For over a century, keyboards and typewriters have endured as the mechanism of choice for text entry. The resiliency of the keyboard, in an era of unprecedented technological change, is the result of how keyboards complement human skills. This may make keyboards difficult to supplant with new input devices or technologies. We summarize some general issues surrounding text entry below, with a focus on mechanical keyboards; see also Lewis et al. [1997].

Skill acquisition and skill transfer — Procedural memory is a specific type of memory that encodes repetitive motor acts. Once an activity is encoded in procedural memory, it requires little conscious effort to perform [Anderson, 1980]. Because procedural memory automates the physical act of text entry, touch-typists can rapidly type words without interfering with the mental composition of text. The process of encoding an activity in procedural memory can be formalized as the *power law of practice*: $T = aP^b$, where T is the time to perform the task, P is the amount of practice, and a and b

are constants that fit the curve to observed data. This suggests that changing the keyboard can have a high relearning cost. However, a change to the keyboard can succeed if it does not interfere with existing skills or allows a significant transfer of skill. For example, some ergonomic keyboards have succeeded by preserving the basic key layout but altering the typing pose to help maintain neutral postures [Honan et al., 1995; Marklin and Simoneau, 1996], whereas the Dvorak key layout may have some small performance advantages but has not found wide adoption due to high retraining costs [Lewis et al., 1997].

Eyes-free operation — With practice, users can memorize the location of commonly used keys relative to the home position of the two hands, allowing typing with little or no visual attention [Lewis et al., 1997]. By contrast, soft keyboards (small on-screen virtual keyboards found on many handheld devices) require nearly constant visual monitoring, resulting in diversion of attention from one's work. Furthermore, with stylus-driven soft keyboards, the user can strike only one key at a time. Thus the design issues for soft keyboards differ tremendously from mechanical keyboards [Zhai et al., 2000].

Tactile feedback — On a mechanical keyboard, users can feel the edges and gaps between the keys, and the keys have an activation force profile that provides feedback of the key strike. In the absence of such feedback, as on touchscreen keyboards [Sears, 1993], performance may suffer and users may not be able to achieve eyes-free performance [Lewis et al., 1997].

Combined text, command, and navigation input — Finally, it is easy to forget that keyboards provide many secondary command and control actions in addition to pure text entry, such as power keys and navigation keys (Enter, Home/End, Delete, Backspace, Tab, Esc, Page Up/Down, arrow keys, etc.), chord key combinations (such as Ctrl+C for Copy) for frequently used commands, and function keys for miscellaneous functions defined by the current application. Without these keys, frequent interleaving of mouse and keyboard activity may be required to perform these secondary functions.

Ergonomic issues — Many modern information workers suffer from repetitive strain injury (RSI). Researchers have identified many risk factors for such injuries, such as working under stress or taking inadequate rest breaks. People often casually associate these problems with keyboards, but the potential for RSI is common to many manually operated tools and repetitive activities [Putz-Anderson, 1988]. Researchers have advocated themes for ergonomic design of keyboards and other devices [Pekelney and Chu, 1995], including reducing repetition, minimizing force required to hold and move the device or to press its buttons, avoiding sharp edges that put pressure on the soft tissues of the hand, and designing for natural and neutral postures of the user's hands and wrists [Honan et al., 1995; Marklin et al., 1997]. Communicating a clear orientation for gripping and moving the device through its industrial design also may help to discourage inappropriate, ergonomically unsound grips.

Other text entry mechanisms — One-handed keyboards can be implemented using simultaneous depression of multiple keys; such *chord keyboards* can sometimes allow one to achieve high peak performance (e.g., court stenographers) but take much longer to learn how to use [Noyes, 1983; Mathias et al., 1996; Buxton, 1990a]. They are often used in conjunction with wearable computers [Smailagic and Siewiorek, 1996] to keep the hands free as much as possible (but see also Section 20.8.1). With complex written languages, such as Chinese and Japanese, key chording and multiple stages of selection and disambiguation are currently necessary for keyboard-based text entry [Wang et al., 2001]. Handwriting and character recognition may ultimately provide a more natural solution, but for Roman languages, handwriting (even on paper, with no recognition involved) is much slower than skilled keyboard use. To provide reliable stylus-driven text input, some systems have adopted unistroke (single-stroke) gestural alphabets [Goldberg and Richardson, 1993] that reduce the demands on recognition technology while remaining relatively easy for users to learn [MacKenzie and Zhang, 1997]. However, small two-thumb keyboards [MacKenzie and Soukoreff, 2002] or fold-away peripheral keyboards are becoming increasingly popular for such devices.

20.8 Modalities of Interaction

Here, we briefly review a number of general strategies and input modalities that have been explored by researchers. These approaches generally transcend a specific type of input device, spanning a range of devices and applications.

20.8.1 Voice and Speech

Carrying on a full conversation with a computer as one might do with another person is well beyond the state of the art today and, even if possible, may be a naïve goal. Yet even without understanding the content of the speech, computers can digitize, store, edit, and replay segments of speech to augment human–human communication [Arons, 1993; Stifelman, 1996]. Conventional voice mail and the availability of MP3 music files on the Web are simple examples of this. Computers can also infer information about the user's activity from ambient audio, such as determining whether the user is present or perhaps engaging in a conversation with a colleague, allowing more timely delivery of information or suppression of notifications that may interrupt the user [Schmandt, 1993; Sawhney and Schmandt, 1999; Horvitz et al., 1999; Buxton, 1995b].

Understanding speech as input is a long-standing area of research. While progress is being made, it is slower than optimists originally predicted, and daunting unsolved problems remain. For limited vocabulary applications with native English speakers, speech recognition can excel at recognizing words that occur in the vocabulary. Error rates can increase substantially when users employ words that are out-of-vocabulary (i.e., words the computer is not "listening" for), when the grammatical complexity of possible phrases increases, or when the microphone is not a high-quality, close-talk headset. Dictation using continuous speech recognition is available on the market today, but the technology still has a long way to go; a recent study found that the corrected words-per-minute rate of text entry using a mouse and keyboard is about twice as fast as dictation input [Karat et al., 1999]. Even if the computer could recognize all of the user's words, the problem of understanding natural language is a significant and unsolved one. It can be avoided by using an artificial language of special commands or even a fairly restricted subset of natural language. Given the current state of the art, however, the closer the user moves toward full, unrestricted natural language, the more difficulties will be encountered.

20.8.2 Pen-Based Gestures and Hand Gesture Input

Pen-based gestures can indicate commands, such as crossing out a word to delete it or circling a paragraph and drawing an arrow to move it. Such gestures support cognitive chunking by integrating command selection with specification of the command's scope [Buxton et al., 1983; Kurtenbach and Buxton, 1991]. Marking menus use directional pen motion to provide extremely rapid menu selection [Kurtenbach and Buxton, 1993; Kurtenbach et al., 1993]. With pen-based interfaces, designers often face a difficult design trade-off between treating the user's marks as ink that is not subject to interpretation vs. providing pen-based input that treats the ink as a potential command [Kramer, 1994; Moran et al., 1997; Mynatt et al., 1999]. Pen input, via sketching, can be used to define 3-D objects [Zeleznik et al., 1996; Igarashi et al., 1999]. Researchers also have explored multimodal pen and voice input; this is a powerful combination because pen and voice have complementary strengths and weaknesses and can disambiguate one another [Cohen et al., 1997; Cohen and Sullivan, 1989; Oviatt, 1997].

Input using hand gestures represents another, less well understood area of inquiry. There are many human behaviors involving hand movements and gestures, but few have been thoroughly explored for human–computer interaction. Cadoz [1994] broadly categorizes hand gestures as semiotic, ergotic, or epistemic. *Semiotic* gestures are those used to communicate meaningful information, such as "thumbs up." *Ergotic* gestures are those used to manipulate physical objects. *Epistemic* gestures are exploratory movements to acquire haptic or tactile information; see also [Kirsh, 1995; Kirsh and Maglio, 1994]. Most research in hand gesture recognition focuses on empty-handed semiotic gestures [Cassell, 2003], which

Rime and Schiaratura further classify as follows [Rime and Schiaratura, 1991]:

Symbolic — Conventional symbolic gestures such as "OK"
Deictic — Pointing to fill in a semantic frame, analogous to deixis in natural language
Iconic — Illustrating a spatial relationship
Pantomimic — Mimicking an invisible tool, such as pretending to swing a golf club

Command input using recognition-based techniques raises a number of unique challenges [Bellotti et al., 2002]. In particular, with most forms of gestural input, errors of user intent and errors of computer interpretation seem inevitable. Deictic gesture in particular has received much attention, with several efforts using pointing (typically captured using instrumented gloves or camera-based recognition) to interact with "intelligent" environments [Baudel and Beaudouin-Lafon, 1993; Maes et al., 1996; Freeman and Weissman, 1995; Jojic et al., 2000]. Deictic gesture in combination with speech recognition has also been studied [Bolt, 1980; Hauptmann, 1989; Lucente et al., 1998; Wilson and Shafer, 2003]. However, there is more to the field than empty-handed semiotic gestures. Recent exploration of tangible interaction techniques [Ishii and Ullmer, 1997] and efforts to sense movements and handling of sensor-enhanced mobile devices perhaps fall more closely under ergotic (manipulative) gestures [Hinckley et al., 2003; Hinckley et al., 2000; Harrison et al., 1998].

20.8.3 Bimanual Input

Aside from touch typing, most of the devices and modes of operation discussed thus far and in use today involve only one hand at a time. But people use both hands in a wide variety of the activities associated with daily life. For example, when writing, a right-hander writes with the pen in the right hand, but the left hand also plays a crucial and distinct role. It holds the paper and orients it to a comfortable angle that suits the right hand. In fact, during many skilled manipulative tasks, Guiard observed that the hands take on asymmetric, complementary roles [Guiard, 1987]: for right-handers, the role of the left hand precedes the right (the left hand first positions the paper), the left hand sets the frame of reference for the action of the right hand (the left hand orients the paper), and the left hand performs infrequent, large-scale movements compared to the frequent, small-scale movements of the right hand (writing with the pen). Most applications for bimanual input to computers are characterized by asymmetric roles of the hands, including compound navigation/selection tasks such as scrolling a Web page and then clicking on a link [Buxton and Myers, 1986], command selection using the nonpreferred hand [Bier et al., 1993; Kabbash et al., 1994], as well as navigation, virtual camera control, and object manipulation in three-dimensional user interfaces [Kurtenbach et al., 1997; Balakrishnan and Kurtenbach, 1999; Hinckley et al., 1998b]. For some tasks, such as banging together a pair of cymbals, the hands may take on symmetric roles. For further discussion of bimanual symmetric tasks, see Guiard, 1987; Balakrishnan and Hinckley, 2000.

20.8.4 Passive Measurement: Interaction in the Background

Not all interaction with computers need consist of explicit, intentionally communicated commands. Think about walking into a modern grocery store. You approach the building; the doors sense this motion and open for you. No explicit communication has occurred, yet a computer has used your action of walking toward the store as an input to decide when to open the door. Intentional, explicit interaction takes place in the foreground, while implicitly sensed interaction takes place in the background, behind the fore of the user's attention [Buxton, 1995a]. Background interaction will be a major emphasis of future research in automation and sensing systems, as users become increasingly mobile and become saturated with information from many sources. Researchers are currently exploring ways to provide context awareness through location sensing, ambient sensing of light, temperature, and other environmental qualities; movement and handling of devices; detection of the user's the identity and physical objects in the environment; and possibly even physiological measures such as heart-rate variability. This type of information potentially can allow technology to interpret the context of a situation and respond more appropriately [Schilit et al., 1994; Schmidt et al., 1999; Dey et al., 2001; Hinckley et al., 2003].

Background interaction can also be applied to explicit input streams through passive behavioral measurements, such as observation of typing speed, manner of moving the cursor, sequence and timing of commands activated in a GUI [Horvitz et al., 1998], and other patterns of use. A carefully designed user interface could make intelligent use of such information to modify its dialogue with the user, based on, for example, inferences about the user's alertness or expertise. These measures do not require additional input devices, but rather gleaning of additional, typically neglected, information from the existing input stream. These are sometimes known as *intelligent* or *adaptive* user interfaces, but mundane examples also exist. For example, cursor control using the mouse or scrolling using a wheel can be optimized by modifying the device response depending on the velocity of movement [Jellinek and Card, 1990; Hinckley et al., 2001].

We must acknowledge the potential for misuse or abuse of information collected in the background. Users should always be made aware of what information is or may potentially be observed as part of a human–computer dialogue. Users should have control and the ability to block any information that they want to remain private [Nguyen and Mynatt, 2001].

20.9 Displays and Perception

We now turn our attention to the fundamental properties of displays and to techniques for effective use of displays. We focus on visual displays and visual human perception, because these represent the vast majority of displays, but we also discuss feedback through the haptic and audio channels.

20.9.1 Properties of Displays and Human Visual Perception

Display requirements, such as resolution in time and space, derive from the properties of human vision. Thus, we begin with the basic issues relating to display brightness, uniformity, and spatial and temporal resolution.

Dynamic range — The human eye has an enormous dynamic range. The amount of light reflected from surfaces on a bright day at the beach is about five orders of magnitude higher than the amount available under dim lamp lights. Yet the shapes, layouts, and colors of objects look nearly identical to the human eye across much of this range. Most displays in common use are self-luminous cathode ray tubes (CRTs) or back-lit liquid crystal displays (LCDs). The best of these devices have a dynamic range (the ratio between the maximum and minimum values produced) of a little more than two orders of magnitude. In practice, under typical room lighting conditions, 15 to 40% of the light reaching the user's eye is actually ambient room light reflected by the front surface of the phosphors, or off the screen surface. This means that the effective dynamic range of most devices, unless viewed in dark rooms, is no better than three or four to one. Fortunately, the human eye can tolerate extreme variation in the overall level of illumination, as well as in the amount of contrast produced by the display.

Spatial frequency — The ability of the human visual system to resolve fine targets is known as *visual acuity*. A standard way to measure visual acuity is to determine how fine a sinusoidal striped pattern can be discriminated from a uniform gray. Humans are capable of perceiving targets as fine as 50 to 60 cycles/degree of visual angle when the pattern is of very high contrast. Figure 20.1 illustrates the spatial sensitivity of the human eye as a function of spatial frequency. Specifically, it illustrates the degree of contrast required for sinusoidal gratings of different spatial frequencies to be perceived. The function has an inverted U shape with a peak at about 2 cycles/degree of visual angle. This means that 5-mm stripes at arm's length are optimally visible. The falloff at low spatial frequencies indicates that the human visual system is insensitive to gradual changes in overall screen luminance. Indeed, most CRTs have a brightness falloff toward the edges of as much as 20%, which we barely notice. This nonuniformity is even more pronounced with rear projection systems, due to the construction of screens that project light primarily in a forward direction. This is called the *screen gain;* a gain of 3.0 means that three times as much light is transmitted in the straight-through

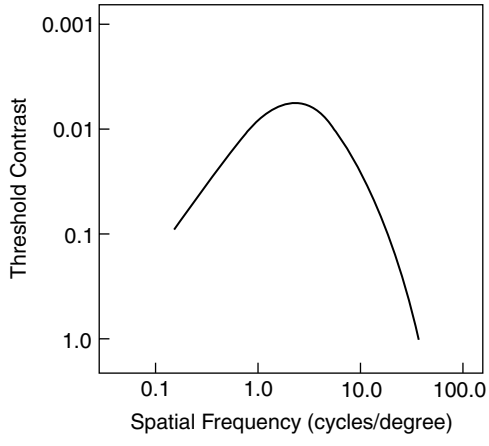

FIGURE 20.1 Spatial contrast sensitivity function of the human visual system. There is a falloff in sensitivity both to detailed patterns (high-spatial frequencies) and to gradually changing gray values (low-spatial frequencies).

direction, compared to a perfect Lambertian diffuser. At other angles, less light is transmitted, so that at a 45° off-axis viewing angle, only half as much light may be available, compared to a perfect diffuser. Screen gain is also available with front projection with similar nonuniformities as a consequence, although the use of curved screens can compensate to some extent.

Spatial resolution — The receptors in the human eye have a visual angle of about 0.8 seconds of arc. Modern displays provide approximately 40 pixels/cm. A simple calculation reveals that at about a 50-cm viewing distance, pixels will subtend about 1.5 seconds of arc, about twice the size of cone receptors in the center of vision. Viewed from 100 cm, such a screen has pixels that will be imaged on the retina at about the same size as the receptors. This might suggest that we are within reach of the perfect display in terms of spatial resolution; such a screen would require approximately 80 pixels/cm at normal viewing distances. However, under some conditions the human visual system is capable of producing superacuities that imply resolution better than the receptor size. For example, during fusion in stereo vision, disparities smaller than 5 seconds of arc can be detected [Westheimer, 1979] (see also Ware, 2000 for discussion of stereopsis and stereo displays aimed at the practitioner). Another example of superacuity is known as *aliasing*, resulting from the division of the screen into discrete pixels; for example, a line on the display that is almost (but not quite) horizontal may exhibit a jagged stairstep pattern that is very noticeable and unsatisfactory. This effect can be diminished by *antialiasing*, which computes pixel color values that are averages of all the different objects that contribute to the pixel, weighted by the percentage of the pixel they cover. Similar techniques can be applied to improve the appearance of text, particularly on LCD screens, where individual red, green, and blue display elements can be used for subpixel antialiasing [Betrisey et al., 2000; Platt, 2000].

Temporal resolution, refresh, and update rates — The *flicker fusion frequency* represents the least rapidly flickering light that the human eye does not perceive as steady. Flicker fusion frequency typically occurs around 50 Hz for a light that turns completely on and off [Wyszecki and Styles, 1982]. In discussing the performance of monitors, it is important to differentiate the refresh rate and the update rate. The *refresh rate* is the rate at which a screen is redrawn, and it is typically constant (values of 60 Hz up to 120 Hz are common). In contrast, the *update rate* is the rate at which the system software updates the output to be refreshed. Ideally, this should occur at or above the refresh rate, but with increasingly demanding applications and complex data sets, this may not be possible. A rule of thumb states that a 10 Hz update rate is a minimum for smooth animation [Robertson et al., 1989]. Motion blur (also known as *temporal antialiasing*) techniques can be applied to reduce the jerky effects resulting from low frame rates [Cook, 1986].

20.10 Color Vision and Color Displays

The single most important fact relating to color displays is that human color vision is trichromatic; our eyes contain three receptors sensitive to different wavelengths. For this reason, it is possible to generate nearly all perceptible colors using only three sets of lights or printing inks. However, it is much more difficult to specify colors exactly using inks than it is using lights, because lights can be treated as a simple vector space, but inks interact in complex nonlinear ways.

20.11 Luminance, Color Specification, and Color Gamut

Luminance is the standard term for specifying brightness, that is, how much light is emitted by a self-luminous display. The luminance system in human vision gives us most of our information about the shape and layout of objects in space. The international standard for color measurement is the CIE (Commission Internationale de l'Eclairage) standard. The central function in Figure 20.2 is the CIE $V(\lambda)$ function, which represents the amount that light of different wavelengths contributes to the overall sensation of brightness. As this curve demonstrates, short wavelengths (blue) and long wavelengths (red) contribute much less than green wavelengths to the sensation of brightness. The CIE tristimulus functions, also shown in Figure 20.2, are a set of color-matching functions that represent the color vision of a typical person. Humans are most sensitive to the green wavelengths around 560 nm. Specifying luminance and specifying a color in CIE tristimulus values are complex technical topics; for further discussion, see Ware, 2000; Wyszecki et al., 1982.

A chromaticity diagram can be used to map out all possible colors perceptible to the human eye, as illustrated in Figure 20.3. The pure spectral hues are given around the boundary of this diagram in nanometers (10^{-9} m). While the spacing of colors in tristimulus coordinates and on the chromaticity diagram is not perceptually uniform, uniform color spaces exist that produce a space in which equal metric distances are closer to matching equal perceptual differences [Wyszecki et al., 1982]. For example, this can be useful to produce color sequences in map displays [Robertson, 1988].

The *gamut* of all possible colors is the dark gray region of the chromaticity diagram, with pure hues at the edge and neutral tones in the center. The triangular region represents the gamut achievable by a particular color monitor, determined by the colors of the phosphors given at the corners of the triangle. Every color within this triangular region is achievable, and every color outside of the triangle is not. This

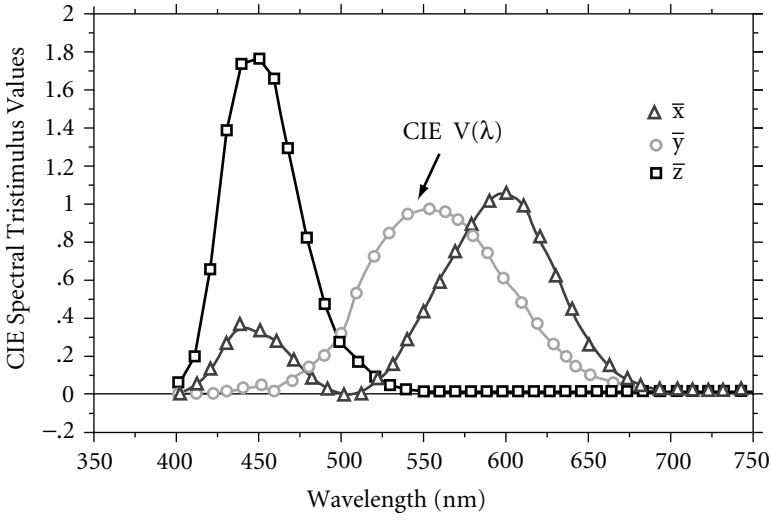

FIGURE 20.2 The CIE tristimulus functions. These are used to represent the standard observer in colorimetry. Short wavelengths at the left-hand side appear blue, in the middle they are green, and to the right they are red. Humans are most sensitive to the green wavelengths around 560 nm.

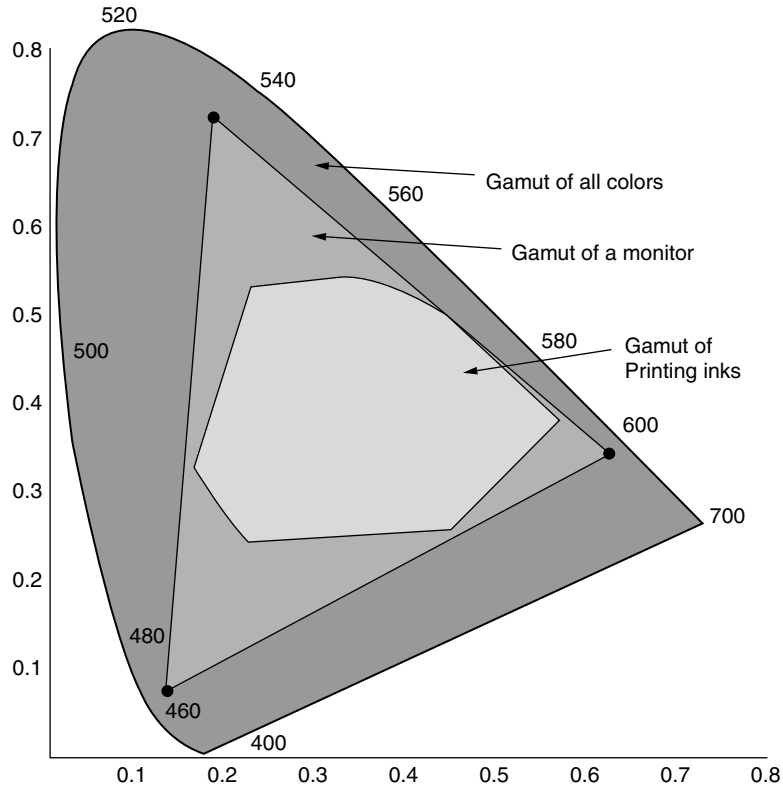

FIGURE 20.3 A CIE chromaticity diagram with a monitor gamut and a printing ink gamut superimposed. It can be seen that the range of available colors with a color printing is smaller than that available with a monitor, and both fall short of providing the full range of color that can be seen.

diagram nicely illustrates the trade-off faced by the designer of color displays. A phosphor that produces a very narrow wavelength band will have chromaticity coordinates close to the pure spectral colors, and this will produce more saturated colors (thus enlarging the triangle). However, this narrow band also means that little light is produced.

The irregular shape inside the triangle illustrates the gamut of colors obtainable using printing inks. Notice that this set of colors is still smaller, causing difficulties when we try to obtain a hard-copy reproduction of the colors on the monitor. Because the eye is relatively insensitive to overall color shifts and overall contrast changes, we can take the gamut from one device and map it into the gamut of the printing inks (or some other device) by compressing and translating it. This is known as *gamut mapping*, a process designed to preserve the overall color relationships while effectively using the range of a device [Stone et al., 1988]. However, it should be noted that the original colors will be lost in this process and that after a succession of gamut mappings, colors may become distorted from their original values.

A process known as *chromatic adaptation* occurs in the human eye receptors and in the early stages of visual processing. For example, we hardly notice that daylight is much bluer than the yellow cast of tungsten light produced from ordinary light bulbs. The CIE standard does not account for chromatic adaptation, nor does it account for color contrast (colors appear differently to the human eye depending on the surrounding visual field). The practical implication is that we can get by with color monitors and printers that are grossly out of calibration. However, accurate color is essential in some applications. It is possible to calibrate a color device precisely so that the particular inputs required to produce a color may be specified in CIE tristimulus values. For monitor calibration, see Cowan, 1983; for calibrating print devices, see Stone et al., 1988. It is also possible to correct for the nonlinear response of CRT displays,

a process known as *gamma correction*, but keep in mind that CRT designers intentionally insert this nonlinearity to match the human eye's sensitivity to relative changes in light intensity. If one desires a set of perceptually equal gray steps, it is usually best to omit gamma correction. See Ware, 2000 for further discussion.

20.12 Information Visualization

Researchers and practitioners have become increasingly interested in communicating large quantities of information quickly and clearly by leveraging the tremendous capabilities of the human visual system, a field known as *information visualization*. Thanks to advances in computer graphics hardware and algorithms, virtually all new desktop machines available today have sophisticated full-color displays with transparency and texture mapping for complex two-dimensional or three-dimensional scenes. It now seems inevitable that these capabilities will become commonplace on laptop computers and ultimately even on handheld devices.

20.12.1 General Issues in Information Coding

The greatest challenge in developing guidelines for information coding is that there are usually effective alternatives, such as color, shape, size, texture, blinking, orientation, and gray value. Although a number of studies compare one or more coding methods separately or in combination, there are so many interactions between the task and the complexity of the display that guidelines based on science are not generally practical. However, Tufte provides excellent guidelines for information coding from an aesthetic perspective [Tufte, ; Tufte, ; Tufte, 1997]. For further discussion, examples, and case studies, see also Ware, 2000 and Card et al., 1999.

A theoretical concept known as *preattentive discrimination* has interesting implications for whether or not the coding used can be processed in parallel by the visual system. The fact that certain coding schemes are processed faster than others is called the *popout* phenomenon, and this is thought to be due to early preattentive processing by the visual system. Thus, for example, the shape of the word *bold* is not processed preattentively, and it will be necessary to scan this entire page to determine how many times the word appears. However, if all of the instances of the word **bold** are emphasized, they pop out at the viewer. This is true as long as there are not too many other emphasized words on the same page: if there are fewer than seven or so instances, they can be processed at a single glance. Preattentive processing is done for color, brightness, certain aspects of texture, stereo disparities, and object orientation and size. Codes that are preattentively discriminable are very useful if rapid search for information is desired [Triesman, 1985]. The following visual attributes are known to be preattentive codes and, therefore, useful in differentiating information belonging to different classes:

Color — Use no more than ten different colors for labeling purposes.
Orientation — Use no more than ten orientations.
Blink coding — Use no more than two blink rates.
Texture granularity — Use no more than five grain sizes.
Stereo depth — The number of depths that can be effectively coded is not known.
Motion — Objects moving out of phase with one another are perceptually grouped. The number of usable phases is not known.

Note that coding multiple dimensions by combining different popout cues is not necessarily effective [Ware, 2000].

20.12.2 Color Information Coding

When considering information display, one of the most important distinctions is between chromatic and luminance information, because these are treated quite differently in human perception. Gray scales are

not perceived in the same way as rainbow-colored scales. A purely chromatic difference is one where two colors of identical luminance, such as red and green, are placed adjacent to one another. Research has shown that we are insensitive to a variety of information if it is presented through purely chromatic changes. This includes shape perception, stereo depth information, shape from shading, and motion. However, chromatic information helps us to classify the material properties of objects. A number of practical implications arise from the differences in the ways luminance information and chromatic information are processed in human vision:

Our spatial sensitivity is lower for chromatic information, allowing image compression techniques to transmit less information about hue relative to luminance.

To make text visible, it is important to make sure that there is a luminance difference between the color of the text and the color of the background. If the background may vary, it is a good idea to put a contrasting border around the letters (e.g., Harrison and Vicente, 1996).

When spatial layout is shown either through a stereo display or through motion cues, ensure adequate luminance contrast.

When fine detail must be shown, for example, with fine lines in a diagram, ensure that there is adequate luminance contrast with the background.

Chromatic codes are useful for labeling objects belonging to similar classes.

Color (both chromatic and gray scale) can be used as a quantitative code, such as on maps, where it commonly encodes height and depth. However, simultaneous contrast effects can change the appearance of a patch of color depending on the surrounding colors; careful selection of colors can minimize this [Ware, 1988].

A number of empirical studies have shown color coding to be an effective way to identify information. It is also effective if used in combination with other cues, such as shape. For example, users may respond to targets faster if the targets can be identified by both shape and color differences (for useful reviews, see Christ, 1975; Stokes et al., 1990; and Silverstein, 1977). Color codes are also useful in the perceptual grouping of objects. Thus, the relationship between a set of different screen objects can be made more apparent by giving them all the same color. However, it is also the case that only a limited number of color codes can be used effectively. The use of more than about ten will cause the color categories to become blurred. In general, there are complex relationships between the type of symbols displayed (e.g., point, line, area, or text), the luminance of the display, the luminance and color of the background, and the luminance and color of the symbol [Spiker et al., 1985].

20.12.3 Integrated Control/Display Objects

When the purpose of a display is to allow a user to integrate diverse pieces of information, it may make sense to integrate the information into a single visual object or *glyph* [Wickens, 1992]. For example, if the purpose is to represent a pump, the liquid temperature could be shown by changing the color of the pump, the capacity could be shown by the overall size of the pump, and the output pressure might be represented by the changing height of a bar attached to the output pipe, rather than a set of individual dials showing these attributes separately. However, perceptual distortions can result from an ill chosen display mapping, and the object display may introduce visual clutter: if there are 50 pumps to control, then the outlines of all the pumps may interfere with the data of interest [Tufte, 1983]. In object displays, input and output can be integrated in a manner analogous to widgets such as the scroll bar, or even more directly by having input devices that resemble the physical object being handled, known as a *prop* [Hinckley et al., 1994a]. For some good examples of the linking of output and input, see Ahlberg and Shneiderman, 1994; Ishii and Ullmer, 1997.

This style of presentation and interaction can be especially relevant for telepresence or augmented reality applications, where the user interacts with actual physical objects with attributes that must be viewed and controlled [Tani et al., 1992; Feiner et al., 1993]. For more abstract data representation tasks, choosing the color, size, orientation, or texture to represent a particular data attribute may be difficult, and there

seem to be practical limits on the number of attributes that one can encode simultaneously. Thus, object displays usually must be custom designed for each display problem. In general, this means that the display and controls should somehow match the user's cognitive model of the task [Norman, 1990; Cole, 1986; Hinckley et al., 1994a].

20.12.4 Three-Dimensional Graphics and Virtual Environments

Much research in three-dimensional information visualization and virtual environments is motivated by the observation that humans naturally operate in physical space and can intuitively move about and remember where things are (an ability known as *spatial memory*). However, translating these potential benefits into artificially generated graphical environments is difficult because of limitations in display and interaction technologies. Virtual environments research pushed this to the limit by totally immersing the user in an artificial world of graphics, but this comes at the cost of visibility and awareness of colleagues and objects in the real world. This has led to research in so-called *fish-tank virtual reality* displays, using a head tracking system in conjunction with a stereo display [Deering, 1992; Ware et al., 1993] or a mirrored setup, which allows superimposition of graphics onto the volume where the user's hands are located [Schmandt, 1983; Serra et al., 1997]. However, much of our ability to navigate without becoming lost depends on the vestibular system and spatial updating as we physically turn our bodies, neither of which is engaged with stationary displays [Loomis et al., 1999; Chance et al., 1998]. For further discussion of navigation in virtual environments, see Darken and Sibert, 1995; Darken and Sibert, 1993. For application of spatial memory to three-dimensional environments, see Robertson et al., 1998; Robertson et al., 1999.

20.12.5 Augmented Reality

Augmented reality superimposes information on the surrounding environment rather than blocking it out. For example, the user may wear a semitransparent display that has the effect of projecting labels and diagrams onto objects in the real world. It has been suggested that this may be useful for training people to use complex systems or for fault diagnosis. For example, a person repairing an aircraft engine could see the names and functions of parts, which could appear superimposed on the parts seen through the display, together with a maintenance record if desired [Caudell and Mizell, 1992; Feiner et al., 1993]. The computer must obtain a detailed model of the environment; otherwise it is not possible to match the synthetic objects with the real ones. Even with this information, correct registration of computer graphics with the physical environment is an extremely difficult technical problem due to measurement error and system latency. This technology has been applied to heads-up displays for fighter aircraft, with semitransparent information about flight paths and various threats in the environment projected on the screen in front of the pilot [Stokes et al., 1990]. It also has been applied to digitally augmented desk surfaces [Wellner, 1993].

20.13 Scale in Displays

It is important to consider the full range of scale for display devices and form-factors that may embody an interaction task. Computing devices increasingly span orders of magnitude in size and computational resources, from watches and handheld personal data assistants (PDAs) to tablet computers and desktop computers, all the way up to multiple-monitor and wall-size displays. A technique that works well on a desktop computer, such as a pull-down menu, may be awkward on a handheld device or even unusable on a wall-size display (where the top of the display may not be within the user's reach). Each class of device seems to raise unique challenges. The best approach may ultimately be to design special-purpose, appliancelike devices (see Want and Borriello, 2000 for a survey) that suit specific purposes.

20.13.1 Small Displays

Users increasingly want to do more and more on handheld devices, mobile phones, pagers, and watches that offer less and less screen real estate. Researchers have investigated various strategies for conserving

screen real estate. Transparent overlays allow divided attention between foreground and background layers [Harrison et al., 1995a; Harrison et al., 1995b; Harrison and Vicente, 1996; Kamba et al., 1996], but some degree of interference seems inevitable. This can be combined with sensing which elements of an interface are being used, such as presenting widgets on the screen only when the user is touching a pointing device [Hinckley and Sinclair, 1999]. Researchers also have experimented with replacing graphical interfaces with graspable interfaces, which respond to tilting, movement, and physical gestures and do not need constant on-screen representations [Fitzmaurice et al., 1995; Rekimoto, 1996; Harrison et al., 1998; Hinckley et al., 2000]. Much research in focus plus context techniques, including fisheye magnification [Bederson, 2000] and zooming metaphors [Perlin and Fox, 1993; Bederson et al., 1996; Smith and Taivalsaari, 1999], has also been motivated by providing more space than the boundaries of the physical screen can provide. Researchers have started to identify principles and quantitative models to analyze the trade-offs between multiple views and zooming techniques [Baudisch et al., 2002; Plumlee and Ware, 2002]. There has been considerable effort devoted to supporting Web browsing in extremely limited screen space (e.g., Buyukkokten et al., 2001; Jones et al., 1999; Trevor et al., 2001).

20.13.2 Multiple Displays

Some very interesting issues arise, however, when multiple displays are considered. Having multiple monitors for a single computer is not like having one large display [Grudin, 2001]. Users employ the boundary between displays to partition their tasks: one monitor is reserved for a primary task, and other monitors are used for secondary tasks. Secondary tasks may support the primary task (e.g., reference material, help files, or floating tool palettes), may provide peripheral awareness of ongoing events (such as an e-mail client) or may provide other background information (to-do lists, calendars, etc.). Switching between applications has a small time penalty (incurred once to switch, and again to return). Perhaps more importantly, it may distract the user or force the user to remember information while switching between applications. Having additional screen space "with a dedicated purpose, always accessible with a glance" [Grudin, 2001] reduces these burdens, and studies suggest that providing multiple, distinct foci for interaction may aid users' memory and recall [Tan et al., 2001; Tan et al., 2002]. Finally, small displays can be used in conjunction with larger displays [Myers et al., 1998; Myers et al., 2000; Rekimoto, 1998], with controls and private information on the small device and shared public information on the larger display. Displays of different dimensions support completely different user activities and social conventions.

20.13.3 Large-Format Displays

Trends in display technology suggest that large-format displays will become increasingly affordable and common. A journal's recent special issue includes numerous articles on implementing large-format displays using projection, application design for large displays, and applications such as automotive design [Funkhouser and Li, 2000]. Large displays often implicitly suggest multiple simultaneous users, with many applications revolving around collaboration [Swaminathan and Sato, 1997; Funkhouser and Li, 2000] and giving a large-scale physical presence to virtual activities [Buxton et al., 2000]. To support input directly on whiteboard-size displays, researchers have explored gestural interaction techniques for pens or touchscreens [Guimbretiere et al., 2001; Moran et al., 1997]. Many technologies cannot handle more than one point of contact, so check this carefully if simultaneous use by multiple persons is desired.

Large displays also seem to lend themselves to interaction at a distance, although using laser pointers to support such interaction [Olsen and Nielsen, 2001] has met with mixed success due to the lack of separation between tracking and dragging states [Buxton, 1990b]. Using handheld devices to interact with the full area of a large display also is problematic, because the ratio of the display to the control surface may be very large [Myers et al., 1998]. Environmentally situated ambient displays share some properties of large displays but emphasize subtle presentation of information in the periphery of attention [Ishii and Ullmer, 1997; Wisneski et al., 1998]. A 3-D environment using multiple large-format

displays that surround the user is known as a *cave* [Cruz-Neira et al., 1993; Buxton and Fitzmaurice, 1998].

Unless life-size viewing of large objects is necessary [Buxton et al., 2000], in general the performance benefits a single large display vs. multiple monitors with the same screen area partitioned by bezels is not yet clear. One recent study suggests that the increased field of view afforded by large-format displays can lead to improved 3-D navigation performance, especially for women [Czerwinski et al., 2002].

20.14 Force and Tactile Displays

Haptic feedback research has sought to provide an additional channel of sensory feedback by synthesizing forces on the skin of the operator. The touch sensation is extraordinarily complex. In fact, the sense of "touch" is a very imprecise term: it includes multiple sensory systems, including sensitivity to pressure, small shear forces in the skin, heat and cold, pain, kinesthesis and proprioception, and the vestibular system [Burdea, 1996].

There appears to be no physical means by which a complex tactile stimulus can be delivered except in a very localized way. As a result, most haptic feedback devices are limited to simulation of a single point of contact, analogous to feeling the world with the tip of a pencil, although a few examples of whole-hand force feedback devices exist [Iwata, 1990; Burdea, 1996]. Efforts in haptic feedback include *force* feedback (active presentation of forces to the user) and *tactile* feedback (active presentation of vibrotactile stimuli to the user). Haptic feedback is popular for gaming devices, such as force feedback steering wheels and joysticks, but general-purpose pointing devices with force or tactile feedback remain uncommon. For a comprehensive discussion of force and tactile feedback technologies and techniques, as well as perceptual properties of the skin and joints, see [Burdea, 1996].

Adding force feedback to a mouse or stylus may impose constraints on the mechanical design, since a physical linkage is typically needed to reflect true forces. This may prevent a force feedback mouse from functioning like a traditional mouse, as it may limit range of motion or preclude clutching by lifting the device. Some devices instead increase resistance between the mouse and the pad, but this prevents simulation of hard contact forces. One can also use a vibrotactile stimulus, such as a vibrating pin under the mouse button or a vibrating shaft in an isometric joystick [Campbell et al., 1999]. Combination devices also have been explored [Akamatsu and Mackenzie, 1996]. Vibrotactile feedback seems especially promising for small mobile devices, for example, to provide the user with feedback of command recognition when the user's attention may not be focused on the screen [Poupyrev et al., 2002]. Applications for remote controls and augmented handles also look promising [Snibbe and MacLean, 2001; MacLean et al., 2000].

Using force feedback to provide attractive forces that pull the user toward a target, or tactile feedback to provide additional feedback for the boundaries of the target, has been found to yield modest speed improvements in some target acquisition experiments, although error rates may also increase [Akamatsu and MacKenzie, 1996; MacKenzie, 1995]. However, there have been almost no published studies about tasks where multiple targets are present, as on a computer screen with many icons and menus. Haptic feedback for one target may interfere with the selection of another, unless one uses techniques such as reducing the haptic forces during rapid motion [Oakley et al., 2001]. Finally, one should also consider whether software constraints, such as snap-to grids, are sufficient to support the user's tasks.

The construction of force output devices is extremely technically demanding. They must be stiff in order to be able to create the sensation of solid contact, yet light so that they have little inertia themselves, and there must be a tight loop between input (position) and output (force). Sigoma [1993] has suggested that having this loop iterate at 5 kHz may be necessary for optimal fine motor control. It has been shown that force feedback improves performance in certain telerobotic applications, such as inserting a peg into a hole [Sheridan, 1992]. The most promising applications of force output seem to appear in applications in which simulation of force is essential, such as surgical simulation and telerobotics [Burdea, 1996].

Another fundamental challenge for haptic feedback techniques results from the interaction between the haptic and visual channels. Visual dominance deals with phenomena resulting from the tendency for vision to dominate other modalities [Wickens, 1992]. Campbell et al. [1999] show that tactile feedback improves

steering through a narrow tunnel, but only if the visual texture matches the tactile texture; otherwise, tactile feedback harms performance.

20.15 Auditory Displays

Here, we consider computer-generated auditory feedback. Audio can consist of synthesized or recorded speech. All other audio feedback is known as *nonspeech audio*. With stereo speakers or a stereo headset, either type of audio can be presented so that it seems to come from a specific 3-D location around the user, known as *spatialized audio*. For speech input and technology-mediated human–human communication applications that treat stored voice as data, see Section 20.8.1.

20.15.1 Nonspeech Audio

Nonspeech auditory feedback is prevalent in video games but largely absent from interaction with computing devices. Providing an auditory echo of the visual interface has little or no practical utility and may annoy users. Audio should be reserved to communicate simple, short messages that complement visual feedback (if any) and will not be referred to later. Furthermore, one or more of the following conditions should hold [Deatherage, 1972; Buxton, 1995b]:

The message should deal with events in time.
The message should call for immediate action.
The message should take place when the user's visual attention may be overburdened or directed elsewhere.

For example, researchers have attempted to enhance scrollbars using audio feedback [Brewster et al., 1994]. However, the meaning of such sounds may not be clear. Gaver advocates ecological sounds that resemble real-world events with an analogous meaning. For example, an empty disc drive might sound like a hollow metal container [Gaver, 1989]. If a long or complex message must be delivered using audio, it will likely be quicker and clearer to deliver it using speech output. Audio feedback may be crucial to support tasks or functionality on mobile devices that must take place when the user is not looking at the display (for some examples, see Hinckley et al., 2000).

Nonspeech sounds can be especially useful for attracting the attention of the user. Auditory alerting cues have been shown to work well, but only in environments where there is low auditory clutter. However, the number of simple nonspeech alerting signals is limited, and this can easily result in misidentification or cause signals to mask one another. An analysis of sound signals in fighter aircraft [Doll and Folds, 1985] found that the ground proximity warning and the angle-of-attack warning on an F16 were both an 800-Hz tone — a dangerous confound because these conditions require opposite responses from the pilot. It can also be difficult to devise nonspeech audio events that convey information without provoking an alerting response that unnecessarily interrupts the user. For example, this design tension arises when considering nonspeech audio cues that convey various properties of an incoming e-mail message [Sawhney and Schmandt, 2000; Hudson and Smith, 1996].

20.15.2 Speech Output

Speech auditory output is generally delivered either through recorded speech segments or completely synthetic speech (also known as *text-to-speech* technology). There has been considerable interest, especially for military applications, in the use of speech in providing warnings to the operators of complex systems. Speech can provide information to direct the operator's attention in a way that alarms cannot; an unfamiliar alarm simply indicates a problem, without telling the user the nature or context of the problem. Synthetic speech is most useful when visual information is not available, for example, in touch-tone telephone menu systems or in screen reader software for blind or low-sighted users. Although progress is being made, synthetic voices still sound somewhat unnatural and may be more difficult for users to understand.

Recorded speech is often used to give applications, particularly games, a more personal feel, but it can only be used for a limited number of responses known in advance.

The rate at which words must be produced to sound natural is a narrow range. For warning messages, 178 words per minute is intelligible but hurried, 123 words per minute is distracting and irritatingly slow, and a more natural rate of 156 words per minute is preferred [Simpson and Marchionda-Frost, 1984]. The playback rate of speech can be increased by overlapping sample times so that one sample is presented to one ear and another sample to the other ear. Technologies to correct for pitch distortions and to remove pauses have also been developed [Arons, 1993; Stifelman, 1996; Sawhney and Schmandt, 2000]. It is recommended by the U.S. Air Force that synthetic speech be 10 dB above ambient noise levels [Stokes et al., 1990].

20.15.3 Spatialized Audio Displays

It is possible to synthesize spatially localized sounds with a quality such that spatial localization in the virtual space is almost as good as localization of sounds in the natural environment [Wenzel, 1992]. Auditory localization appears to be primarily a two-dimensional phenomenon. That is, observers can localize, to some degree of accuracy, in horizontal position (*azimuth*) and elevation angle. Azimuth and elevation accuracies are of the order of 15°. As a practical consequence, this means that sound localization is of little use in identifying sources in conventional screen displays. Where localized sounds are really useful is in providing an orienting cue or warning about events occurring behind the user, outside of the field of vision.

There is also a well known phenomenon called *visual capture of sound*. Given a sound and an apparent visual source for the sound, such as a talking face on a cinema screen, the sound is perceived to come from the apparent source despite the fact that the actual source may be off to one side. Thus, visual localization tends to dominate auditory localization when both kinds of cues are present.

20.16 Future Directions

The future of interaction with computers will be both very different and very much like it is today. Some of our current tools, such as mice and keyboards, have evolved to suit interaction with desktop GUIs and rapid text entry. As long as users' work continues to involve desktop computers and text entry, we will continue to see these devices in use — not only because they are familiar, but also because they closely match human skills and the requirements of the task. Thus, devising new techniques that provide more efficient pointing at a display than a mouse, for example, is difficult to achieve [Card et al., 1978]. Speech recognition will allow new types of interaction and may enable interaction where it previously has been difficult or infeasible. However, we will continue to interact with computers using our hands and physical intermediaries, not necessarily because our technology requires us to do so, but because touching, holding, and moving physical objects is the foundation of the long evolution of tool use in the human species [Wilson, 1998].

But our computers and the tasks they serve are rapidly evolving. Current handheld devices have the display and computational capabilities of common desktop machines from ten years ago. What is lacking is new methods of interacting with such devices that uniquely suit mobile interaction, rather than derivatives of the desktop interface. Researchers are still actively exploring and debating the best ways to achieve this. Meanwhile, technology advances and economic trends continue to drive the cost of commodity displays lower and lower, while the limits of the technology continue to increase. Thus, we will continue to see new innovations in both very small and very large displays. As these become commonplace, new forms of interaction will become prevalent. Very small displays invariably seem to be incorporated into input/output appliances such as watches, pagers, and handheld devices, so interaction techniques for very small form-factors will become increasingly important.

The Internet and wireless networking seem to be the main disruptive technologies of the current era. Indeed, it seems likely that 100 years from now the phrase "wireless network" will seem every bit as antiquated as the phrase "horseless carriage" does today. Nobody really understands yet what it will mean for everything and everyone to be connected, but many researchers are working to explore the vision

of ubiquitous computing originally laid out by Mark Weiser [Weiser, 1991]. Techniques that allow users to communicate and share information will become increasingly important. Biometric sensors or other convenient means for establishing identity will make services such as personalization of the interface and data sharing much simpler [Rekimoto, 1997; Sugiura and Koseki, 1998]. Techniques that combine dissimilar input devices and displays in interesting ways also will be important to realize the full potential of these technologies (e.g., Myers et al., 2001; Streitz et al., 1999). Electronic tagging techniques for identifying objects [Want et al., 1999] may also become commonplace. Such diversity of locations, users, and task contexts points to the increasing importance of sensors to acquire contextual information, as well as machine learning techniques to interpret them and infer meaningful actions [Buxton, 1995a; Bellotti et al., 2002; Hinckley et al., 2003]. This may well lead to an age of ubiquitous sensors [Saffo, 1997] with devices that can see, feel, and hear through digital perceptual mechanisms.

References

Accot, J., and S. Zhai (1997). Beyond Fitts' law: Models for trajectory-Based HCI tasks. *Proc. CHI '97: ACM Conference on Human Factors in Computing Systems*. 295–302.

Accot, J., and S. Zhai (1999). Performance evaluation of input devices in trajectory-based tasks: An application of the steering law. *Proc. CHI '99*, Pittsburgh, PA. 466–472.

Accot, J., and S. Zhai (2001). Scale effects in steering law tasks. *Proc. CHI 2001 ACM Conference on Human Factors in Computing Systems*. 1–8.

Accot, J., and S. Zhai (2002). More than dotting the i's — foundations for crossing-based interfaces. *ACM CHI 2002 Conf. on Human Factors in Computing Systems*. 73–80.

Ahlberg, C., and B. Shneiderman (1994). The alphaslider: a compact and rapid selector. *CHI '94*. 365–371.

Akamatsu, M., and I.S. Mackenzie (1996). Movement characteristics using a mouse with tactile and force feedback. *International Journal of Human–Computer Studies* **45**: 483–493.

Anderson, J.R. (1980). Chapter 8: Cognitive Skills. *Cognitive Psychology and Its Implications*. San Francisco, W. H. Freeman: 222–254.

Arons, B. (1993). SpeechSkimmer: interactively skimming recorded speech. *UIST '93 Symp. on User Interface Software and Technology*. 187–195.

Balakrishnan, R., T. Baudel, G. Kurtenbach, and G. Fitzmaurice (1997). The rockin' mouse: integral 3-D manipulation on a plane. *CHI '97 Conf. on Human Factors in Computing Systems*. 311–318.

Balakrishnan, R., and K. Hinckley (1999). The role of kinesthetic reference frames in two-handed input performance. *Proc. ACM UIST '99 Symp. on User Interface Software and Technology*. 171–178.

Balakrishnan, R., and K. Hinckley (2000). Symmetric bimanual interaction. *Proc. ACM CHI 2000 Conference on Human Factors in Computing Systems*. 33–40.

Balakrishnan, R., and G. Kurtenbach (1999). Exploring bimanual camera control and object manipulation in 3-D graphics interfaces. *Proc. CHI '99 ACM Conf. on Human Factors in Computing Systems*. 56–63.

Balakrishnan, R., and I.S. MacKenzie (1997). Performance differences in the fingers, wrist, and forearm in computer input control. *Proc. CHI '97 ACM Conf. on Human Factors in Computing Systems*. 303–310.

Baudel, T., and M. Beaudouin-Lafon (1993). Charade: Remote control of objects using hand gestures. *Communications of the ACM* **36**(7): 28–35.

Baudisch, P., N. Good, V. Bellotti, and P. Schraedley (2002). Keeping things in context: a comparative evaluation of focus plus context screens, overviews, and zooming. *CHI 2002 Conf. on Human Factors in Computing Systems*. 259–266.

Bederson, B. (2000). Fisheye menus. *Proc. ACM UIST 2000 Symposium on User Interface Software and Technology*. 217–226.

Bederson, B., J. Hollan, K. Perlin, J. Meyer, D. Bacon and G. Furnas (1996). Pad++: a zoomable graphical sketchpad for exploring alternate interface physics. *Journal of Visual Languages and Computing* **7**:3–31.

Bellotti, V., M. Back, W.K. Edwards, R. Grinter, C. Lopes, and A. Henderson (2002). Making sense of sensing systems: five questions for designers and researchers. *Proc. ACM CHI 2002 Conference on Human Factors in Computing Systems*. 415–422.

Betrisey, C., J. Blinn, B. Dresevic, B. Hill, G. Hitchcock, B. Keely, D. Mitchell, J. Platt, and T. Whitted (2000). Displaced filtering for patterned displays. *Proc. Society for Information Display Symposium*. 296–299.

Bier, E., M. Stone, K. Pier, W. Buxton, and T. DeRose (1993). Toolglass and magic lenses: the see-through interface. *Proceedings of SIGGRAPH 93*, Anaheim, CA. 73–80.

Bolt, R. (1980). Put-that-there: voice and gesture at the graphics interface. *Computer Graphics* (Aug.): 262–270.

Brewster, S.A., P.C. Wright, and A.D.N. Edwards (1994). The design and evaluation of an auditory-enhanced scrollbar. *Proc. ACM CHI '94 Conference Proceedings on Human Factors in Computing Systems*. 173–179.

Britton, E., J. Lipscomb, and M. Pique (1978). Making nested rotations convenient for the user. *Computer Graphics* **12**(3): 222–227.

Brooks, J., F.P. (1988). Grasping reality through illusion: interactive graphics serving science. *Proceedings of CHI '88: ACM Conference on Human Factors in Computing Systems*, Washington, DC, ACM, New York. 1–11.

Bukowski, R., and C. Sequin (1995). Object associations: A simple and practical approach to virtual 3-D manipulation. *ACM 1995 Symposium on Interactive 3-D Graphics*. 131–138.

Burdea, G. (1996). *Force and Touch Feedback for Virtual Reality*. New York, John Wiley and Sons.

Buxton, B., and G. Fitzmaurice (1998). HMDs, caves and chameleon: a human-centric analysis of interaction in virtual space. *Computer Graphics* **32**(8): 69–74.

Buxton, W. (1983). Lexical and pragmatic considerations of input structure. *Computer Graphics* **17**(1): 31–37.

Buxton, W. (1986). Chunking and phrasing and the design of human–computer dialogues. *Information Processing '86, Proc. of the IFIP 10th World Computer Congress*, Amsterdam, North Holland Publishers. 475–480.

Buxton, W. (1990a). The pragmatics of haptic input. *Proceedings of CHI '90: ACM Conference on Human Factors in Computing Systems, Tutorial 26 Notes*, Seattle, WA, ACM, New York.

Buxton, W. (1990b). A three-state model of graphical input. *Proc. INTERACT '90*, Amsterdam, Elsevier Science. 449–456.

Buxton, W. (1995a). Integrating the periphery and context: a new taxonomy of telematics. *Proceedings of Graphics Interface '95*. 239–246.

Buxton, W. (1995b). Speech, language and audition. *Readings in Human–Computer Interaction: Toward the Year 2000*, R. Baecker, J. Grudin, W. Buxton, and S. Greenberg, Eds. San Francisco, Morgan Kaufmann Publishers. 525–537.

Buxton, W., G. Fitzmaurice, R. Balakrishnan, and G. Kurtenbach (2000). Large displays in automotive design. *IEEE Computer Graphics and Applications* (July/August): 68–75.

Buxton, W., E. Fiume, R. Hill, A. Lee, and C. Woo (1983). Continuous hand-gesture driven input. *Proceedings of Graphics Interface '83*. 191–195.

Buxton, W., R. Hill, and P. Rowley (1985). Issues and techniques in touch-sensitive tablet input. *Computer Graphics* **19**(3): 215–224.

Buxton, W., and B. Myers (1986). A study in two-handed input. *Proceedings of CHI '86: ACM Conference on Human Factors in Computing Systems*, Boston, MA, ACM, New York. 321–326.

Buyukkokten, O., H. Garcia-Molina, and A. Paepcke (2001). Accordion summarization for end-game browsing on PDAs and cellular phones. *ACM CHI 2001 Conf. on Human Factors in Computing Systems*, Seattle, WA. 213–220.

Cadoz, C. (1994). *Les réalités virtuelles*. Dominos, Flammarion.

Campbell, C., S. Zhai, K. May, and P. Maglio (1999). What you feel must be what you see: adding tactile feedback to the trackpoint. *Proceedings of INTERACT '99: 7th IFIP conference on Human–Computer Interaction*. 383–390.

Card, S., W. English, and B. Burr (1978). Evaluation of mouse, rate-controlled isometric joystick, step keys, and text keys for text selection on a CRT. *Ergonomics* **21**: 601–613.

Card, S., J. Mackinlay, and G. Robertson (1991). A morphological analysis of the design space of input devices. *ACM Transactions on Information Systems* **9**(2): 99–122.

Card, S., J. Mackinlay, and B. Shneiderman (1999). *Readings in Information Visualization: Using Vision to Think*. San Francisco, Morgan Kaufmann.

Cassell, J. (2003). A framework for gesture generation and interpretation. In *Computer Vision in Human-Machine Interaction*, R. Cipolla and A. Pentland, Eds. Cambridge, Cambridge University Press. (In press.)

Caudell, T.P., and D.W. Mizell (1992). Augmented reality: an application of heads-up display technology to manual manufacturing processes. *Proc. HICSS '92*. 659–669.

Chance, S., F. Gaunet, A. Beall, and J. Loomis (1998). Locomotion mode affects the updating of objects encountered during travel: The contribution of vestibular and proprioceptive inputs to path integration. *Presence* **7**(2): 168–178.

Chen, M., S.J. Mountford, and A. Sellen (1988). A study in interactive 3-D rotation using 2-D control devices. *Computer Graphics* **22**(4): 121–129.

Cholewiak, R., and A. Collins (1991). Sensory and physiological bases of touch. *The Psychology of Touch*, M. Heller and W. Schiff, Eds. Hillsdale, NJ, Lawrence Erlbaum. 23–60.

Christ, R.E. (1975). Review and analysis of color coding research for visual displays. *Human Factors* **25**: 71–84.

Cohen, P., M. Johnston, D. McGee, S. Oviatt, J. Pittman, I. Smith, L. Chen, and J. Clow (1997). QuickSet: multimodal interaction for distributed applications. *ACM Multimedia 97*. 31–40.

Cohen, P.R., and J.W. Sullivan (1989). Synergistic use of direct manipulation and natural language. *Proc. ACM CHI '89 Conference on Human Factors in Computing Systems*. 227–233.

Cole, W.G. (1986). Medical cognitive graphics. *ACM CHI '86 Conf. on Human Factors in Computing Systems*. 91–95.

Conner, D., S. Snibbe, K. Herndon, D. Robbins, R. Zeleznik, and A. van Dam (1992). Three-dimensional widgets. *Computer Graphics (Proc. 1992 Symposium on Interactive 3-D Graphics)*. 183–188, 230–231.

Cook, R.L. (1986). Stochastic sampling in computer graphics. *ACM Trans. Graphics* **5**(1): 51–72.

Cowan, W.B. (1983). An inexpensive calibration scheme for calibrations of a color monitor in terms of CIE standard coordinates. *Computer Graphics* **17**(3): 315–321.

Cruz-Neira, C., D. Sandin, and T. DeFanti (1993). Surround-screen projection-based virtual reality: The design and implementation of the CAVE. *Computer Graphics (SIGGRAPH Proceedings)*. 135–142.

Czerwinski, M., D. S. Tan, and G. G. Robertson (2002). Women take a wider view. *Proc. ACM CHI 2002 Conference on Human Factors in Computing Systems*, Minneapolis, MN. 195–202.

Darken, R.P., and J.L. Sibert (1993). A toolset for navigation in virtual environments. *Proc. ACM UIST '93. Symposium on User Interface Software and Technology.* 157–165.

Darken, R.P., and J.L. Sibert (1995). Navigating large virtual spaces. *International Journal of Human–Computer Interaction* (Oct.).

Deatherage, B.H. (1972). Auditory and other sensory forms of information presentation. In *Human Engineering Guide to Equipment Design*, H. Van Cott and R. Kinkade, Eds. U.S. Government Printing Office.

Deering, M. (1992). High-resolution virtual reality. *Computer Graphics* **26**(2): 195–202.

Dey, A., G. Abowd, and D. Salber (2001). A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Journal of Human–Computer Interaction* **16**(2–4): 97–166.

Doll, T.J., and D.J. Folds (1985). Auditory signals in military aircraft: ergonomic principles versus practice. *Proc. 3rd Symp. Aviation Psych*, Ohio State University, Dept. of Aviation, Colombus, OH. 111–125.

Douglas, S., A. Kirkpatrick, and I. S. MacKenzie (1999). Testing pointing device performance and user assessment with the ISO 9241, part 9 standard. *Proc. ACM CHI '99 Conf. on Human Factors in Computing Systems*. 215–222.

Douglas, S., and A. Mithal (1994). The effect of reducing homing time on the speed of a finger-controlled isometric pointing device. *Proc. ACM CHI '94 Conf. on Human Factors in Computing Systems*. 411–416.

Feiner, S., B. Macintyre, and D. Seligmann (1993). Knowlege-based augmented reality. *Communications of the ACM* **36**(7): 53–61.

Fitzmaurice, G., and W. Buxton (1997). An empirical evaluation of graspable user interfaces: toward specialized, space-multiplexed input. *Proceedings of CHI '97: ACM Conference on Human Factors in Computing Systems*, Atlanta, GA, ACM, New York. 43–50.

Fitzmaurice, G., H. Ishii, and W. Buxton (1995). Bricks: laying the foundations for graspable user interfaces. *Proceedings of CHI '95: ACM Conference on Human Factors in Computing Systems*, Denver, CO, ACM, New York. 442–449.

Fitzmaurice, G.W., R. Balakrishnan, and G. Kurtenbach (1999). Sampling, synthesis, and input devices. *Commun. ACM* **42**(8): 54–63.

Foley, J.D., V. Wallace, and P. Chan (1984). The human factors of computer graphics interaction techniques. *IEEE Computer Graphics and Applications* (Nov.): 13–48.

Freeman, W.T., and C. Weissman (1995). Television control by hand gestures. *Intl. Workshop on Automatic Face and Gesture Recognition*, Zurich, Switzerland. 179–183.

Funkhouser, T., and K. Li (2000). Large format displays. *IEEE Comput. Graphics Appl.* (July–Aug. special issue): 20–75.

Gaver, W. (1989). The SonicFinder: an interface that uses auditory icons. *Human–Computer Interaction* **4**(1): 67–94.

Gibson, J. (1962). Observations on active touch. *Psychological Review* **69**(6): 477–491.

Gibson, J. (1986). *The Ecological Approach to Visual Perception*. Hillsdale, NJ, Lawrence Erlbaum Assoc.

Goldberg, D., and C. Richardson (1993). Touch-typing with a stylus. *Proc. INTERCHI '93 Conf. on Human Factors in Computing Systems*. 80–87.

Grudin, J. (2001). Partitioning digital worlds: focal and peripheral awareness in multiple monitor use. *Proc. ACM CHI 2001 Conference on Human Factors in Computing Systems*. 458–465.

Guiard, Y. (1987). Asymmetric division of labor in human skilled bimanual action: the kinematic chain as a model. *The Journal of Motor Behavior* **19**(4): 486–517.

Guiard, Y., F. Buourgeois, D. Mottet, and M. Beaudouin-Lafon (2001). Beyond the 10-bit barrier: Fitts' law in multi-scale electronic worlds. *People and Computers XV: Interaction with on Fractions. Joint Proc. IHM 2001 and HCI 2001*. Springs. 573–581.

Guimbretiere, F., M.C. Stone, and T. Winograd (2001). Fluid interaction with high-resolution wall-size displays. *Proc. UIST 2001 Symp. on User Interface Software and Technology*. 21–30.

Harrison, B., K. Fishkin, A. Gujar, C. Mochon, and R. Want (1998). Squeeze me, hold me, tilt me! An exploration of manipulative user interfaces. *Proc. ACM CHI '98 Conf. on Human Factors in Computing Systems*. 17–24.

Harrison, B., H. Ishii, K. Vicente, and W. Buxton (1995a). Transparent layered user interfaces: an evaluation of a display design to enhance focused and divided attention. *Proceedings of CHI '95: ACM Conference on Human Factors in Computing Systems*. 317–324.

Harrison, B., G. Kurtenbach, and K. Vicente (1995b). An experimental evaluation of transparent user interface tools and information content. *Proc. ACM UIST '95*. 81–90.

Harrison, B., and K. Vicente (1996). An experimental evaluation of transparent menu usage. *Proceedings of CHI '96: ACM Conference on Human Factors in Computing Systems*. 391–398.

Hauptmann, A. (1989). Speech and gestures for graphic image manipulation. *Proceedings of CHI '89: ACM Conference on Human Factors in Computing Systems*, Austin, TX, ACM, New York. 241–245.

Hinckley, K., E. Cutrell, S. Bathiche, and T. Muss (2001). Quantitative analysis of scrolling techniques. *Proc. ACM CHI 2002. Conf. on Human Factors in Computing Systems.* 65–72.

Hinckley, K., M. Czerwinski, and M. Sinclair (1998a). Interaction and modeling techniques for desktop two-handed input. *Proceedings of the ACM UIST '98 Symposium on User Interface Software and Technology*, San Francisco, CA, ACM, New York. 49–58.

Hinckley, K., R. Pausch, J. Goble, and N. Kassell (1994a). Passive real-world interface props for neurosurgical visualization. *Proceedings of CHI '94: ACM Conference on Human Factors in Computing Systems*, Boston, MA, ACM, New York. 452–458.

Hinckley, K., R. Pausch, J.C. Goble, and N.F. Kassell (1994b). A survey of design issues in spatial input. *Proceedings of the ACM UIST '94 Symposium on User Interface Software and Technology*, Marina del Rey, CA, ACM, New York. 213–222.

Hinckley, K., R. Pausch, D. Proffitt, and N. Kassell (1998b). Two-handed virtual manipulation. *ACM Transactions on Computer–Human Interaction* **5**(3): 260–302.

Hinckley, K., J. Pierce, E. Horvitz, and M. Sinclair (2003). Foreground and background interaction with sensor-enhanced mobile devices. *ACM TOCHI*. Special issue on sensor-based interaction (to appear).

Hinckley, K., J. Pierce, M. Sinclair, and E. Horvitz (2000). Sensing techniques for mobile interaction. *ACM UIST 2000 Symp. on User Interface Software and Technology*. 91–100.

Hinckley, K., and M. Sinclair (1999). Touch-sensing input devices. *ACM CHI '99 Conf. on Human Factors in Computing Systems*. 223–230.

Hinckley, K., M. Sinclair, E. Hanson, R. Szeliski, and M. Conway (1999). The VideoMouse: A camera-based multi-degree-of-freedom input device. *ACM UIST '99 Symp. on User Interface Software and Technology*. 103–112.

Hinckley, K., J. Tullio, R. Pausch, D. Proffitt, and N. Kassell (1997). Usability analysis of 3-D rotation techniques. *Proc. ACM UIST '97 Symp. on User Interface Software and Technology*, Banff, Alberta, Canada, ACM, New York. 1–10.

Hix, D., J. Templeman, and R. Jacob (1995). Pre-screen projection: from concept to testing of a new interaction technique. *Proc. ACM CHI '95*. 226–233.

Honan, M., E. Serina, R. Tal, and D. Rempel (1995). Wrist postures while typing on a standard and split keyboard. *Proc. HFES Human Factors and Ergonomics Society 39th Annual Meeting*. 366–368.

Horvitz, E., J. Breese, D. Heckerman, D. Hovel, and K. Rommelse (1998). The Lumiere Project: Bayesian user modeling for inferring the goals and needs of software users. *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, Madison, WI, San Francisco, Morgan Kaufmann. 256–265.

Horvitz, E., A. Jacobs, and D. Hovel (1999). Attention-sensitive alerting. *Proceedings of UAI '99, Conference on Uncertainty and Artificial Intelligence.* 305–313.

Hudson, S., and I. Smith (1996). Electronic mail previews using non-speech audio. *CHI '96 Companion Proceedings*. 237–238.

Igarashi, T., S. Matsuoka, and H. Tanaka (1999). Teddy: a sketching interface for 3-D freeform design. *ACM SIGGRAPH '99*, Los Angeles, CA. 409–416.

Ishii, H., and B. Ullmer (1997). Tangible bits: Toward seamless interfaces between people, bits, and atoms. *Proceedings of CHI '97: ACM Conference on Human Factors in Computing Systems*, Atlanta, GA, ACM, New York. 234–241.

Iwata, H. (1990). Artificial reality with force-feedback: development of desktop virtual space with compact master manipulator. *Computer Graphics* **24**(4): 165–170.

Jacob, R. (1991). The use of eye movements in human–computer interaction techniques: what you look at is what you get. *ACM Transactions on Information Systems* **9**(3): 152–169.

Jacob, R., L. Sibert, D. McFarlane, and M. Mullen, Jr. (1994). Integrality and separability of input devices. *ACM Transactions on Computer–Human Interaction* **1**(1): 3–26.

Jellinek, H., and S. Card (1990). Powermice and user performance. *Proc. ACM CHI '90 Conf. on Human Factors in Computing Systems*. 213–220.

Jojic, N., B. Brumitt, B. Meyers, and S. Harris (2000). Detecting and estimating pointing gestures in dense disparity maps. *Proceed. of IEEE Intl. Conf. on Automatic Face and Gesture Recognition*. 468.

Jones, M., G. Marsden, N. Mohd-Nasir, K. Boone, and G. Buchanan (1999). Improving Web interaction on small displays. *Computer Networks* **31**(11–16): 1129–1137.

Kabbash, P., W. Buxton, and A. Sellen (1994). Two-handed input in a compound task. *Proceedings of CHI '94: ACM Conference on Human Factors in Computing Systems*, Boston, MA, ACM, New York. 417–423.

Kamba, T., S.A. Elson, T. Harpold, T. Stamper, and P. Sukaviriya (1996). Using small screen space more efficiently. *Conference Proceedings on Human Factors in Computing Systems*. 383.

Karat, C., C. Halverson, D. Horn, and J. Karat (1999). Patterns of entry and correction in large vocabulary continuous speech recognition systems. *Proc. ACM CHI '99 Conf. on Human Factors in Computing Systems*. 568–575.

Kirsh, D. (1995). Complementary strategies: why we use our hands when we think. *Proceedings of 7th Annual Conference of the Cognitive Science Society.* Hillsdale, NJ, Lawrence Erlbaum. 212–217.

Kirsh, D., and P. Maglio (1994). On distinguishing epistemic from pragmatic action. *Cognitive Science* **18**(4): 513–549.

Kramer, A. (1994). Translucent patches — dissolving windows. *Proc. ACM UIST '94 Symp. on User Interface Software and Technology*. 121–130.

Kurtenbach, G., and W. Buxton (1991). Issues in combining marking and direct manipulation techniques. *Proc. UIST '91*. 137–144.

Kurtenbach, G., and W. Buxton (1993). The limits of expert performance using hierarchic marking menus. *Proc. INTERCHI '93*. 482–487.

Kurtenbach, G., G. Fitzmaurice, T. Baudel, and B. Buxton (1997). The design of a GUI paradigm based on tablets, two-hands, and transparency. *Proceedings of CHI '97: ACM Conference on Human Factors in Computing Systems*, Atlanta, GA, ACM, New York. 35–42.

Kurtenbach, G., A. Sellen, and W. Buxton (1993). An emprical evaluation of some articulatory and cognitive aspects of "marking menus." *Journal of Human–Computer Interaction* **8**(1).

Lewis, J., K. Potosnak, and R. Magyar (1997). Keys and keyboards. In *Handbook of Human–Computer Interaction*, M. Helander, T. Landauer, and P. Prabhu, Eds., Amsterdam, North-Holland. 1285–1316.

Lipscomb, J., and M. Pique (1993). Analog input device physical characteristics. *SIGCHI Bulletin* **25**(3): 40–45.

Loomis, J., R.L. Klatzky, R.G. Golledge, and J.W. Philbeck (1999). Human navigation by path integration. In *Wayfinding: Cognitive Mapping and Other Spatial Processes*, R.G. Golledge, Ed. Baltimore, Johns Hopkins. 125–151.

Lucente, M., G. Zwart, and A. George (1998). Visualization space: a testbed for deviceless multimodal user interface. *AAAI '98*.

MacKenzie, I.S. (1992). Fitts' law as a research and design tool in human-computer interaction. *Human–Computer Interaction* **7**: 91–139.

MacKenzie, I.S. (1995). Input devices and interaction techniques for advanced computing. In *Virtual Environments and Advanced Interface Design*, W. Barfield and T. Furness, Eds. Oxford, Oxford University Press. 437–470.

MacKenzie, I.S., T. Kauppinen, and M. Silfverberg (2001). Accuracy measures for evaluating computer pointing devices. *CHI 2001*. 9–16.

MacKenzie, I.S., and A. Oniszczak (1998). A comparison of three selection techniques for touchpads. *Proc. ACM CHI '98 Conf. on Human Factors in Computing Systems*. 336–343.

MacKenzie, I.S., A. Sellen, and W. Buxton (1991). A comparison of input devices in elemental pointing and dragging tasks. *Proc. ACM CHI '91 Conf. on Human Factors in Computing Systems*. 161–166.

MacKenzie, I.S., and R.W. Soukoreff (2002). A model of two-thumb text entry. *Proceedings of Graphics Interface*, Toronto, Canadian Information Processing Society. 117–124.

MacKenzie, I.S., and C. Ware (1993). Lag as a determinant of human performance in interactive systems. *Proc. ACM INTERCHI '93*. 488–493.

MacKenzie, I.S., and S. Zhang (1997). The immediate usability of graffiti. *Proc. Graphics Interface '97*. 129–137.

MacLean, K.E., S.S. Snibbe, and G. Levin (2000). Tagged handles: merging discrete and continuous control. *Proc. ACM CHI 2000 Conference on Human Factors in Computing Systems*, The Hague, Netherlands.

Maes, P., T. Darrell, B. Blumberg, and A. Pentland (1996). The ALIVE system: wireless, full-body interaction with autonomous agents. *ACM Multimedia Systems* (Special Issue on Multimedia and Multisensory Virtual Worlds).

Marklin, R., and G. Simoneau (1996). Upper extremity posture of typists using alternative keyboards. *ErgoCon '96*. 126–132.

Marklin, R., G. Simoneau, and J. Monroe (1997). The effect of split and vertically inclined computer keyboards on wrist and forearm posture. *Proc. HFES Human Factors and Ergonomics Society 41st Annual Meeting*. 642–646.

Mathias, E., I.S. MacKenzie, and W. Buxton (1996). One-handed touch typing on a QWERTY keyboard.*Human–Computer Interaction* **11**(1): 1–27.

McGuffin, M., and R. Balakrishnan (2002). Acquisition of expanding targets. *CHI Letters* **4**(1).

Mine, M., F. Brooks, and C. Sequin (1997). Moving objects in space: exploiting proprioception in virtual-environment interaction. *Computer Graphics* **31**(Proc. SIGGRAPH '97): 19–26.

Moran, T., P. Chiu, and W. van Melle (1997). Pen-based interaction techniques for organizing material on an electronic whiteboard. *Proc. ACM UIST '97 Symp. on User Interface Software and Technology*. 45–54.

Myers, B., K. Lie, and B. Yang (2000). Two-handed input using a PDA and a mouse. *CHI 2000*. 41–48.

Myers, B., R. Miller, C. Evankovich, and B. Bostwick (2001). Individual use of hand-held and desktop computers simultaneously. (Submitted).

Myers, B., H. Stiel, and R. Gargiulo (1998). Collaboration using multiple PDAs connected to a PC. *Proc. ACM CSCW '98 Conf. on Computer Supported Cooperative Work*, Seattle, WA. 285–294.

Mynatt, E.D., T. Igarashi, W.K. Edwards, and A. LaMarca (1999). Flatland: new dimensions in office whiteboards. *ACM SIGCHI Conference on Human Factors in Computing Systems*, Pittsburgh, PA. 346–353.

Nguyen, D.H., and E. Mynatt (2001). Toward visibility of a ubicomp environment.

Norman, D. (1990). *The Design of Everyday Things*. New York, Doubleday.

Noyes, J. (1983). Chord keyboards. *Applied Ergonomics* **14**: 55–59.

Oakley, I., S. Brewster, and P. Gray (2001). Solving multi-target haptic problems in menu interaction. *Proc. ACM CHI 2001 Conf. on Human Factors in Computing Systems: Extended Abstracts*. 357–358.

Olsen, D.R., and T. Nielsen (2001). Laser pointer interaction. *Proc. ACM CHI 2001 Conf. on Human Factors in Computing Systems*. 17–22.

Oviatt, S. (1997). Multimodal interactive maps: Designing for human performance. *Human–Computer Interaction* **12**: 93–129.

Pearson, G., and M. Weiser (1988). Exploratory evaluation of a planar foot-operated cursor-positioning device. *Proc. ACM CHI '88 Conference on Human Factors in Computing Systems*. 13–18.

Pekelney, R., and R. Chu (1995). Design criteria of an ergonomic mouse computer input device. *Proc. HFES Human Factors and Ergonomics Society 39th Annual Meeting*. 369–373.

Perlin, K., and D. Fox (1993). Pad: an alternative approach to the computer interface. *SIGGRAPH '93*.

Platt, J. (2000). Optimal filtering for patterned displays. *IEEE Signal Processing Letters* **7**(7): 179–83.

Plumlee, M., and C. Ware (2002). Modeling performance for zooming vs. multi-window interfaces based on visual working memory. *AVI '02: Advanced Visual Interfaces*, Trento Italy.

Poupyrev, I., S. Maruyama, and J. Rekimoto (2002). Ambient touch: designing tactile interfaces for handheld devices. *UIST 2002 Symp. on User Interface Software and Technology*. 51–60.

Putz-Anderson, V. (1988). *Cumulative Trauma Disorders: a Manual for Musculoskeletal Diseases of the Upper Limbs*. Bristol, PA, Taylor and Francis.

Rekimoto, J. (1996). Tilting operations for small screen interfaces. *Proc. ACM UIST '96*. 167–168.

Rekimoto, J. (1997). Pick-and-drop: a direct manipulation technique for multiple computer environments. *Proc. ACM UIST '97*. 31–39.

Rekimoto, J. (1998). A multiple device approach for supporting whiteboard-based interactions. *CHI '98*. 344–351.

Rime, B., and L. Schiaratura (1991). Gesture and speech. In *Fundamentals of Nonverbal Behavior*, R.S. Feldman and B. Rimé, Eds., New York, Press Syndicate of the University of Cambridge. 239–281.

Robertson, G., M. Czerwinski, K. Larson, D. Robbins, D. Thiel, and M. van Dantzich (1998). Data Mountain: using spatial memory for document management. *UIST '98*.

Robertson, G., M. van Dantzich, D. Robbins, M. Czerwinski, K. Hinckley, K. Risden, V. Gorokhovsky, and D. Thiel (1999). The task gallery: a 3-D window manager. To appear in *ACM CHI 2000*.

Robertson, G.G., S.K. Card, and J.D. Mackinlay (1989). The cognitive coprocessor architecture for interactive user interfaces. *Proc. UIST '89 Symposium on User Interface Software and Technology*. 10–18.

Robertson, P.K. (1988). Perceptual color spaces. Visualizing color gamuts: a user interface for the effective use of perceptual color spaces in data display. *IEEE Comput. Graphics Appl.* **8**(5): 50–64.

Rutledge, J., and T. Selker (1990). Force-to-motion functions for pointing. *Proc. of Interact '90: The IFIP Conf. on Human–Computer Interaction*. 701–706.

Saffo, P. (1997). Sensors: The next wave of infotech innovation. *Institute for the Future: 1997 Ten-Year Forecast*. 115–122.

Sawhney, N., and C. Schmandt (1999). Nomadic radio: scaleable and contextual notification for wearable audio messaging. *CHI '99*. 96–103.

Sawhney, N., and C.M. Schmandt (2000). Nomadic radio: speech and audio interaction for contextual messaging in nomadic environments. *ACM Transactions on Computer–Human Interaction* **7**(3): 353–383.

Schilit, B.N., N.I. Adams, and R. Want (1994). Context-aware computing applications. *Proc. IEEE Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, IEEE Computer Society. 85–90.

Schmandt, C.M. (1983). Spatial input/display correspondence in a stereoscopic computer graphic work station. *Computer Graphics (Proc. ACM SIGGRAPH '83)* **17**(3): 253–262.

Schmandt, C.M. (1993). From desktop audio to mobile access: opportunities for voice in computing. *Advances in Human–Computer Interaction*, H.R. Hartson and D. Hix, Eds. Norwood, NJ, Ablex. 251–283.

Schmidt, A., K. Aidoo, A. Takaluoma, U. Tuomela, K. Van Laerhove, and W. Van de Velde (1999). Advanced interaction in context. *Handheld and Ubiquitous Computing (HUC'99)*. Heidelberg, Springer-Verlag: 89–101.

Sears, A. (1993). Investigating touchscreen typing: the effect of keyboard size on typing speed. *Behaviour and Information Technology* **12**(1): 17–22.

Sears, A., C. Plaisant, and B. Shneiderman (1992). A new era for high precision touchscreens. *Advances in Human–Computer Interaction* **3:** 1–33.

Sears, A., and B. Shneiderman (1991). High precision touchscreens: design strategies and comparisons with a mouse. *International Journal of Man–Machine Studies* **34**(4): 593–613.

Sellen, A., G. Kurtenbach, and W. Buxton (1992). The prevention of mode errors through sensory feedback. *Human–Computer Interaction* **7**(2): 141–164.

Serra, L., N. Hern, C. Beng Choon, and T. Poston (1997). Interactive vessel tracing in volume data. *ACM/SIGGRAPH Symposium on Interactive 3-D Graphics*, Providence, RI, ACM, New York. 131–137.

Sheridan, T.B. (1992). *Telerobotics, Automation, and Human Supervisory Control*. Cambridge, MA, MIT Press.

Sigoma, K.B. (1993). A survey of perceptual feedback issues in dexterous telemanipulation: part I. Finger force feedback. *Proc. IEEE Virtual Reality Annu. Int. Symp.* 263–270.

Silverstein, D. (1977). Human factors for color display systems. *Color and the Computer: Concepts, Methods and Research*. New York, Academic Press. 27–61.

Simpson, C.A., and K. Marchionda-Frost (1984). Synthesized speech rate and pitch effects on intelligibility of warning messages for pilots. *Human Factors* **26**: 509–517.

Smailagic, A., and D. Siewiorek (1996). Modalities of interaction with CMU wearable computers. *IEEE Personal Communications* (Feb.): 14–25.

Smith, R.B., and A. Taivalsaari (1999). Generalized and stationary scrolling. *Proc. UIST '99: CHI Letters* **1**(1): 1–9.

Snibbe, S., and K. MacLean (2001). Haptic techniques for media control. *CHI Letters (Proc. UIST 2001)* **3**(2): 199–208.

Spiker, A., S. Rogers, and J. Cicinelli (1985). Selecting color codes for a computer-generated topographic map based on perception experiments and functional requirements. *Proc. 3rd Symp. Aviation Psychology*, Ohio State University, Dept. of Aviation, Columbus, OH. 151–158.

Stifelman, L. (1996). Augmenting real-world objects: a paper-based audio notebook. *CHI '96 Conference Companion*. 199–200.

Stokes, A., C. Wickens, and K. Kite (1990). *Display Technology: Human Factors Concepts*. Warrendale, PA, SAE.

Stone, M.C., W.B. Cowan, and J.C. Beatty (1988). Color gamut mapping and the printing of digital color images. *ACM Trans. Graphics* **7**(4): 249–292.

Stratton, G. (1897). Vision without inversion of the retinal image. *Psychological Review* **4**: 360–361.

Streitz, N.A., J. Geiβler, T. Holmer, S. Konomi, C. Müller-Tomfelde, W. Reischl, P. Rexroth, R.P. Seitz, and Steinmetz (1999). i-LAND: an interactive landscape for creativity and innovation. *ACM CHI '99 Conf. on Human Factors in Computing Systems*, Pittsburgh, PA. 120–127.

Sugiura, A., and Y. Koseki (1998). A user interface using fingerprint recognition — holding commands and data objects on fingers. *UIST '98 Symp. on User Interface Software and Technology*. 71–79.

Sutherland, I.E., (1968). A head-mounted three-dimensional display. *Proc. the Fall Joint Computer Conference*. 757–764.

Swaminathan, K., and S. Sato (1997). Interaction design for large displays. *Interactions* (Jan–Feb).

Tan, D.S., J.K. Stefanucci, D.R. Proffitt, and R. Pausch (2001). The infocockpit: providing location and place to aid human memory. *Workshop on Perceptive User Interfaces*, Orlando, FL.

Tan, D.S., J.K. Stefanucci, D.R. Proffitt, and R. Pausch (2002). Kinesthesis aids human memory. *CHI 2002 Extended Abstracts*, Minneapolis, MN.

Tani, M., K. Yamaashi, K. Tanikoshi, M. Futakawa, and S. Tanifuji (1992). Object-oriented video: interaction with real-world objects through live video. *Proceedings of ACM CHI '92 Conference on Human Factors in Computing Systems*. 593–598, 711–712.

Trevor, J., D.M. Hilbert, B.N. Schilit, and T.K. Koh (2001). From desktop to phonetop: a UI for Web interaction on very small devices. *Proc. UIST '01 Symp. on User Interface Software and Technology*. 121–130.

Triesman, A. (1985). Preattentive processing in vision. *Comput. Vision, Graphics and Image Process* **31**: 156–177.

Tufte, E.R. (1983). *The Visual Display of Quantitative Information*. Cheshire, CT, Graphics Press.

Tufte, E.R. (1990). *Envisioning Information*. Cheshire, CT, Graphics Press.

Tufte, E.R. (1997). *Visual Explanations: Images and Quantities, Evidence and Narrative*. Cheshire, CT, Graphics Press.

Wang, J., S. Zhai, and H. Su (2001). Chinese input with keyboard and eye-tracking: an anatomical study. *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*.

Want, R., and G. Borriello (2000). Survey on information appliances. *IEEE Personal Communications* (May/June): 24–31.

Want, R., K.P. Fishkin, A. Gujar, and B.L. Harrison (1999). Bridging physical and virtual worlds with electronic tags. *Proc. ACM CHI '99 Conf. on Human Factors in Computing Systems*. 370–377.

Ware, C. (1988). Color sequences for univariate maps: theory, experiments, and principles. *IEEE Comput. Graphics Appl.* **8**(5): 41–49.

Ware, C. (2000). *Information Visualization: Design for Perception*. San Francisco, Morgan Kaufmann.

Ware, C., K. Arthur, and K. S. Booth (1993). Fish tank virtual reality. *Proceedings of ACM INTERCHI '93 Conference on Human Factors in Computing Systems*. 37–41.

Ware, C., and J. Rose (1999). Rotating virtual objects with real handles. *ACM Transactions on CHI* **6**(2): 162–180.

Weiser, M. (1991). The computer for the 21st century. *Scientific American* (Sept.): 94–104.

Wellner, P. (1993). Interacting with paper on the DigitalDesk. *Communications of the ACM* **36**(7): 87–97.

Wenzel, E.M. (1992). Localization in virtual acoustic displays. *Presence* **1**(1): 80–107.

Westheimer, G. (1979). Cooperative nerual processes involved in stereoscopic acuity. *Exp. Brain Res.* **36**: 585–597.

Wickens, C. (1992). *Engineering Psychology and Human Performance*. New York, HarperCollins.

Wilson, A., and S. Shafer (2003). XWand: UI for intelligent spaces. *CHI 2003*. To appear.

Wilson, F. R. (1998). *The Hand: How Its Use Shapes the Brain, Language, and Human Culture*. New York, Pantheon Books.

Wisneski, C., H. Ishii, A. Dahley, M. Gorbet, S. Brave, B. Ullmer, and P. Yarin (1998). Ambient displays: turning architectural space into an interface between people and digital information. *Lecture Notes in Computer Science* **1370**: 22–32.

Wyszecki, G., and W.S. Styles (1982). *Color Science, 2nd Ed.* New York, Wiley.

Zeleznik, R., K. Herndon, and J. Hughes (1996). SKETCH: an interface for sketching 3-D scenes. *Proceedings of SIGGRAPH '96*, New Orleans, LA. 163–170.

Zhai, S. (1998). User performance in relation to 3-D input device design. *Computer Graphics* **32**(8): 50–54.

Zhai, S., M. Hunter, and B.A. Smith (2000). The Metropolis keyboard — an exploration of quantitative techniques for virtual keyboard design. *CHI Letters* **2**(2): 119–128.

Zhai, S., and P. Milgram (1993). Human performance evaluation of isometric and elastic rate controllers in a 6DoF tracking task. *Proc. SPIE Telemanipulator Technology*.

Zhai, S., P. Milgram, and W. Buxton (1996). The influence of muscle groups on performance of multiple degree-of-freedom input. *Proceedings of CHI '96: ACM Conference on Human Factors in Computing Systems*, Vancouver, British Columbia, Canada, ACM, New York. 308–315.

Zhai, S., C. Morimoto, and S. Ihde (1999). Manual and gaze input cascaded (MAGIC) pointing. *Proc. ACM CHI '99 Conf. on Human Factors in Computing Systems*. 246–253.

Zhai, S., B.A. Smith, and T. Selker (1997). Improving browsing performance: a study of four input devices for scrolling and pointing tasks. *Proc. INTERACT '97: The Sixth IFIP Conf. on Human–Computer Interaction*. 286–292.

21

Secondary Storage Systems

21.1 Introduction
Roadmap to the Chapter
21.2 Single Disk Organization and Performance
Disk Organization • Disk Arm Scheduling • Methods to Improve Disk Performance
21.3 RAID Disk Arrays
Motivation for RAID • RAID Concepts • RAID Fault-Tolerance and Classification • Caching and the Memory Hierarchy • RAID Reliability Modeling
21.4 RAID1 or Mirrored Disks
Request Scheduling with Mirrored Disks • Scheduling of Write Requests with an NVS Cache • Mirrored Disk Layouts
21.5 RAID5 Disk Arrays
RAID5 Operation in Normal Mode • RAID5 Operation in Degraded Mode • RAID5 Operation in Rebuild Mode
21.6 Performance Evaluation Studies
Single Disk Performance • RAID Performance • Analysis of RAID5 Systems
21.7 Data Allocation and Storage Management in Storage Networks
Requirements of a Storage Network • Storage Management
21.8 Conclusions and Recent Developments

Alexander Thomasian
New Jersey Institute of Technology

21.1 Introduction

The magnetic disk technology was developed in the 1950s to provide a higher capacity than magnetic drums, which with one head per track constituted a *fixed head* system, while disks introduced a *movable head* system. The movable heads were shared by all tracks, resulting in a significant cost reduction. IBM's *RAMAC — Random Access Method for Accounting and Control* [Matick 1977] — announced in 1956 was the first magnetic disk storage system with a capacity of 5 Megabytes (MB) and a $10,000 price tag.

Magnetic disks have been a multibillion industry for many years. There has been a recent dramatic decrease in the cost per megabyte of disk capacity, with a resulting drop in disk prices. This is a result of a rapid increase in disk capacity: 62% *CAGR (compound annual growth rate)* [Wilkes 2003], which is due to increases in areal recording density.

The explosion in storage capacity in computer installations has led to a dramatic increase in the cost of storage management. In 1980, one data administrator was in charge of 10 GB (gigabytes) of data, while in

2000 this number was 1 TB (terabyte) [Gray 2002]. Given that 5 TB costs $5000, the cost of administering it is an order of magnitude higher. Consequently, automating storage management and data allocation has emerged as a very important area [Wilkes 2003].

Despite the importance of disks and secondary storage, there has been relatively little discussion about them in the computer science and engineering literature. This is due to the fact that more attention has been paid traditionally to the **central processing unit (CPU)**, which determines the speed of the computation. This situation changed significantly with the publication of a paper on **RAID (Redundant Array of Independent Disks)** [Patterson et al. 1998]. More recently there has been an explosion in the level of research activity in storage systems.

In fact, there has been a constant stream of articles dealing with scheduling and performance analysis of disks and disk subsystems. Magnetic bubbles and charged couple devices (CCDs) were considered candidates for replacing disks, or at least serving as an intermediate level storage between disks and main memory, but neither of the two prophecies became a reality. Some other technologies, such as Flash memories, optical disks, and magnetic tapes, are reviewed in Hennessey and Patterson [2003].

DRAM memories, which are *volatile* because they lose their contents when power is removed, were projected to replace disks a decade ago (see Section 2.4 in Gibson [1992]). **Nonvolatile storage (NVS)** with DRAM (NVRAM) can be realized by means of *UPS (uninterruptible power supply)* and it has been argued that a duplexed NVS has about the same reliability as disk storage. This allows **fast writes**, that is, the writing of data to disk is considered completed as soon as the data is written onto duplexed NVS [Menon and Cortney 1993].

The longevity of disk storage with respect to DRAM is attributable to the lower cost of storage per megabyte by a factor of 1000 to one, the higher recording density, and also the nonvolatility.

The nonvolatility feature is quite important and *micro-electromechanical system*-**(MEMS) based storage**, which is nonvolatile and an order of magnitude faster than disks is a technology contending to replace disks, although it is currently ten times more expensive [Uysal et al. 2003]. MEMS-based storage differs from disks in that data can be accessed by moving the read/write heads in two dimensions, while magnetic disks are rotated (angularly) and the read-write heads are moved radially to access the data. Another interesting technology is **magnetic RAM (MRAM)**, which has approximately DRAM speeds but is less costly, stores 100s MB/chip, and is nonvolatile [Wilkes 2003]. It will take time for these technologies to replace disks, but fundamental principles of storage hierarchies are expected to prevail. Given the current dominance of magnetic disks and disk arrays, the current chapter solely concerns itself with this technology.

The impact of magnetic disks on computing is comparable to magnetic core memories, whose introduction resulted in a dramatic increase in feasible memory sizes. Programs that encountered fewer interruptions due to page faults, because their working sets could reside in main memory, could run much faster (see Chapter 85 and Chapter 86).

When the main memory sizes were small, only one program could be held in main memory and when that program stopped — due to (1) completion, (2) preemption because its time quantum expired, or (3) a page fault or an I/O request — switchover (context switching) to another program was slow. Switchover required the workspace of the current program to be swapped out onto a drum or a disk (this is not required in case (1)) before the next program can be loaded into main memory. In fact, this time-critical and frequent swapping activity was carried out onto drums, even after disks became available, because drums, unlike disks, do not incur a seek time and are hence faster. Swapping occurs less frequently in current systems, which utilize disks, whose performance (especially seek time) has improved considerably, so that disks have completely replaced drums.

The context switching overhead, which took tens of milliseconds, was eliminated when larger memories were introduced. Current DRAM memories are in gigabytes, while the largest and expensive core memories were in megabytes. Consequently, there is less need to swap out processes and, furthermore, several programs can be held in main memory, so that the CPU can switch from one program to another at a small cost. *Multiprogramming*, even when a single processor is available, results in an increased system throughput, which is due to the overlap in CPU and I/O processing.

Shared memory multiprocessors (SMPs) allow multiple programs residing in main memory to execute concurrently. This necessitates more disks and I/O paths to handle the higher rate of disk accesses.

The space held by user and system programs is quite small compared to the space held by data, so that it will not be considered further in our discussions. We are mainly concerned with accesses to datasets associated with *DBMSs (database management systems)*, such as the tables of an operational relational DBMS. In fact, *data warehouses*, which are utilized by *decision support systems (DSS)* [Ramakrishnan and Gehrke 2003] occupy more space than operational databases.

In this chapter we do not concern ourselves with the structure of file systems, and the metadata associated with them, because it is covered in Chapter 86 on *Secondary Storage and Filesystems*.

Magnetic disks made the transition from **batch processing** to **OLTP (online transaction processing)** possible. As an example of batch processing, consider a banking application. Debit and credit transactions originating from bank tellers were entered onto punched cards, which were then transferred to a magnetic tape (direct document entry onto tape or disk became possible later). Transactions were sorted by customer number before updating the master file (M/F), which was already in that order. A new M/F tape with an updated balance was created by means of a daily run of the batch update program. The batch update process was (1) error prone (due to manual data entry), (2) costly because it was labor intensive, (3) slow in that the updating was carried out once daily, and (4) risky in that overdrafts were a possibility.

To check the balance of a customer record in the M/F required accessing half of the file records on the average. It would have been impossible to implement relational DBMSs without the fast random access capability provided by disks.

Magnetic disks usually update data in place, while magnetic tapes write modified data sequentially onto another tape, which makes data compression/decompression possible. Data compression with in-place updating does not work with disks because modified data may have a lower compression ratio than the original data, so that the compressed data might not fit into its original space any longer. Compression in disks can be used in conjunction with the *log-structured file system (LFS)* paradigm, which is discussed in Section 21.3.3.

The time to access a record on a disk file depends on the file organization [Ramakrishnan and Gehrke 2003]. A *sequential scan* of an unordered files or a **heap file** is $O(N)$ and the cost is $N/2$, on average, if the record is found and N otherwise. A *binary search* on a sorted file with N records can be used to find the desired record in $log_2 N$ steps. **Hashed files** apply a hashing function to a *search field* to map a record into a bucket (e.g., a 4- or 8-KB disk block). The record can be retrieved with one disk access, provided that the bucket has not overflowed. *Linear hashing* and *extendible hashing* are two methods to deal with bucket overflows and underflows (empty buckets) when there are dynamic insertions and deletions [Ramakrishnan and Gehrke 2003].

B+ trees are the most popular indexing structure utilized in database management systems. The number of accesses for a file indexed with a B+ tree index is one access (for the page containing the desired record) plus the number of levels traversed in the index. The top two levels of the B+ tree, which typically has three or four levels, are usually cached, thus reducing the number of disk accesses by two. Given an indexed or hashed file organization, OLTP in the form of online updating of a customer record is possible, while the customer is at an *ATM (automatic teller machine)*. More importantly, balances can be checked for withdrawals and overdrafts can be prevented.

Despite this relatively short disk access time (less than 10 milliseconds), I/O time might be a significant contributor to application response time. With the advent of very high-speed CPUs, transaction response time for OLTP applications is determined by the number of disk accesses resulting from database buffer *misses*. A miss means that the requested page cannot be found in the database buffer, as opposed to a *hit*, when it is. The *miss ratio* is used to quantify this effect. The maximum throughput of an OLTP benchmark can be designated as the point where the 95th percentile of transaction response time exceeds 2 seconds.

Reducing disk access time is also important when users are involved in an online interaction with a computer system, such as program development, CAD/CAM, etc. Studies have shown that reducing computer response time for interactive applications results in a synergistic reduction in user think time. More specifically, for *subsecond computer response times*, user reaction time is faster because the user

operates from short-term memory; consequently, there is a reduction in the overall duration of the activity [Hennessey and Patterson 2003]. More generally, disk access time affects the end-to-end response time and *Quality-of-Service (QoS)* for many applications.

21.1.1 Roadmap to the Chapter

In Section 21.2, we describe the organization of a single disk in some detail and then provide a rather comprehensive review of disk scheduling paradigms.

What makes an array of disks more interesting than a *just a bunch of disks (JBOD)* is that using them as an ensemble provides additional capabilities (e.g., for parallel access and/or to protect data against data failures. In Section 21.3 we discuss general concepts associated with RAID. In Section 21.4 and Section 21.5 we discuss the two most important RAID levels: RAID1 (mirrored disks) and RAID5 (rotated parity arrays). Performance evaluation studies of disks and disk arrays are discussed in Section 21.6.

Issues associated with data allocation and storage management in storage networks are discussed in Section 21.7. In Section 21.8, after summarizing the discussion in the chapter, we discuss topics that are not covered here, but are expected to gain importance in the future.

21.2 Single Disk Organization and Performance

Two disk performance metrics of interest to this discussion are the **data transfer rate** and the **mean access time**. The *sustained data transfer rate*, rather than the *instantaneous data transfer rate*, is of interest. The sustained rate is smaller than the instantaneous rate because of the inter-record gaps, the check block,* and disk addressing information separating data blocks (this is also to identify faulty sectors). The number of sectors per track and hence the disk transfer rate can be increased considerably by adopting the *noID sector format*, where the header information is stored in solid-state memory rather than on the disk surface. In modern disk drives, the data transfer rate from outer tracks is higher than inner tracks due to zoning, which is explained below.

The mean access time ($\bar{x}_{disk}$) is the metric of interest when disk accesses are to randomly placed blocks of data, as in OLTP applications. The maximum number of requests satisfied by the disk per second is the reciprocal of $\bar{x}_{disk}$, assuming a FCFS scheduling policy, although much higher throughputs can be obtained by the scheduling policies described later in this section.

The **disk positioning time**, or the time to place the R/W head at the beginning of the block being accessed, dominates the disk service time when small disk blocks are being accessed. The positioning time is the sum of seek time to move the arm and rotational latency until the desired data block reaches the R/W head. Early disk scheduling methods dealt with minimizing seek time, while it is minimizing the positioning time that provides the best performance.

In fact, the best way to reduce disk access time is to eliminate disk access altogether! This can be accomplished by caching and prefetching at any one of the levels of the memory hierarchy. The **five-minute rule** states: "Keep a data item in main memory if its access frequency is five minutes or higher, otherwise keep it in magnetic memory" [Gray and Reuter 1993]. Of course, the parameters of this rule change in time.

This section is organized as follows. In Section 21.2.1 we describe the organization of a modern disk drive. In Section 21.2.2 we review disk arm scheduling techniques for requests to discrete and continuous data (typically used in multimedia applications) and combinations of the two. In Section 21.2.3 we discuss various other methods to improve disk performance, which include reorganizing the data layout, etc.

*Things are more complicated than this. There is a first line of protection for a sector, a second line of protection for a group of sectors, and a third line of protection for blocks of blocks of sectors.

21.2.1 Disk Organization

A disk drive consists of one or more **platters** or disks on which data is recorded in concentric circular **tracks**. The platters are attached to a spindle, which rotates the platters at a **constant angular velocity (CAV)**, which means that the linear velocity is higher on outer tracks.*

There are a set of read/write **(R/W) heads** attached to an **actuator** or **disk arm**, where each head can read/write from a track on one side of a platter. At any time instant, the R/W heads can only access tracks located on the same disk **cylinder**, but only one R/W head is active at any time. The time to activate another R/W head is called the **track switching time**.

The arm can be moved to access any of the C disk cylinders and the time required to move the arm is called the **seek time**. The seek from the outermost to innermost cylinder, referred to as a **full-stroke** seek, has the maximum seek time, which is less than 10 milliseconds (ms) for modern disk drives. Cylinder-to-cylinder seeks tend to take less time than head switching time (both are under 1 ms). The **seek time characteritic** $t(d)$, $1 \leq d \leq C-1$ is an increasing function of the seek distance d, but usually a few irregularities are observed in $t(d)$, which is measured by averaging repeated seeks with the same seek distances.

Modern disks accelerate the arm to some maximum speed, let it coast at the speed it has attained, and then decelerate the arm until it stops. Applying curve-fitting to measurement data yields the seek time $t(d)$ vs. the seek distance d. A representative function is $t(d) = a + b\sqrt{d}$ for $1 \leq d \leq d_0$ and $t(d) = e + fd$ for $d_0 \leq d \leq C-1$, where d_0 designates the beginning of the linear portion of the seek.

When the blocks being accessed are uniformly distributed over all the blocks of a non-zoned disk (a disk with the same number of blocks per cylinder), the number of accesses over the C cylinders of the disk will be uniform. It follows from a geometrical probability argument that the average distance for a random seek is one third the disk cylinders: $\overline{d} \approx C/3$. Because the seek time characteristic $(t(d), 1 \leq d \leq C-1)$ of the disk is nonlinear, the mean seek time $\bar{x}_{seek}$ is not equal to $t(\bar{d})$, but rather $\overline{x}_{seek} = \sum_{d=1}^{C-1} P[d]t(d)$, where $P[d]$ is the probability that the seek distance is d.

The seek distance probability mass function, assuming that the probability of no seek is p and the other cylinders are accessed uniformly, is given by: $P[0] = p$ and $P[d] = 2(1-p)(C-d)/[C(C-1)], 1 \leq d \leq C-1$. To derive the density function, we note that there are two seeks of distance $C-1$ and $2(C-d)$ seeks of distance d. The normalization factor is $\sum_{d=1}^{C-1} 2(C-d) = C(C-1)$. For uniform accesses to all cylinders, $P[1] = 1/C$ and $P[d] = 2(C-d)/C^2, 1 \leq d \leq C-1$. Expressions for $P[d]$ in zoned disks are given in Thomasian and Menon [1997].

In addition to radial positioning of R/W heads to access the desired block, angular positioning is accomplished by rotating the disk at a fixed velocity, (e.g., 7200 rotations per minute [RPM]). The delay to access data, called **rotational latency**, is uniformly distributed over disk rotation time T_{rot}. The average latency is $T_{rot}/2$, that is, 4.17 ms for 7200 RPM. There have been suggestions to place duplicate data blocks [Zhang et al. 2002] or provide two access arms [Ng 1991], 180° apart in both cases.

It is best to write data files sequentially, that is, on consecutive sectors of a track, consecutive tracks of a cylinder, consecutive cylinders, so that sequential reads can be carried out efficiently. In fact, the layout of sectors has been optimized for this purpose by introducing **track skew** and **cylinder skew** to mask head and cylinder switching times. The first sector on a succeeding track (or cylinder) is skewed to match the head switching time (or a single cylinder seek), so that no additional latency is incurred in reading sequential data.

There has been a rapid increase in areal magnetic disk recording density due to an increase in the number of **tracks per inch (TPI)** and the number of **bits per inch (BPI)** on a track. A typical areal recording density of 35 Gigabits per square inch was possible at the end of the 20th century [Gray and Shenoy 2000] and a continued increase in this density has been projected, so that 100 Gigabits per square inch seems possible.

*CD-ROMs (compact disc read-only memory) adjust the speed of the motor so that the linear speed is always constant; hence, **constant linear velocity (CLV)**.

Obviously, the effective recording density is lower than the raw recording density. The increase in areal density has led to small form-factor (3.5-inch diameter) disks, whose capacity exceeds 100 Gigabytes (GB), while 500-GB disks have been projected to be available in a few years. Higher TPI has resulted in a reduction in the seek distance and seek time, but this is at the cost of head-settling time [Ng 1998]. Disk seek time remains a major contributor to positioning time.

Almost all disks follow the **Fixed Block Architecture (FBA)** format, as opposed to the variable block size **count-key-data (CKD)** format, where the (physical) block size is determined by application requirements. FBA organizes data as 512-byte blocks called **sectors**, which are addressable by their block number. Each sector is protected by an **ECC (error correcting code)** for error detection and correction, which can detect error bursts in tens of bits and correct several bytes in a sector. Such information is available in disk product manuals, which are retrievable online (see, e.g., the Maxtor Atlas 15K).

Roll-mode or **zero-latency** read (and write) capability allows the sectors of a block to be read out-of-order, that is, starting with any sector. When the data transfer starts from the middle of the block, the latency equals the time of a full rotation time (T_{rot}) minus the transfer time of a block ($\bar{x}_{block}$). The average latency is the weighted sum of two cases: (1) with probability $q = \#\_sector\_per\_block/\#\_sectors\_per\_track$, the R/W head lands inside the block to be read, in which case the average latency is $\bar{x}_{latency} = T_{rot} + \bar{x}_{sector}/2$; and (2) with probability $1 - q$, the head lands outside the block, so that $\bar{x}_{latency} = (T_{rot} - \bar{x}_{block})/2$. Given that q and $\bar{x}_{block}$ are negligibly small for smaller block sizes, it follows that $\bar{x}_{latency} \approx T_{rot}/2$. This capability reduces the latency considerably when reading larger blocks of data (e.g., reading a full track).

Until a decade ago, most disks stored a fixed number of bits on all disk tracks, so that all tracks had a fixed transfer rate. This resulted in a lower recording density than was technologically possible at outer disk tracks (with a longer circumference than inner tracks), thus resulting in a lower overall disk capacity.

Disk zoning or **zoned constant angular velocity (ZCAV)** or **zoned bit recording (ZBR)** or **notches** or **notched drives** (used in the SCSI standard) were introduced to deal with this inefficiency. The number of bits per track, and consequently the number of sectors per track in such disks, is increased from innermost to outermost tracks, keeping the linear recording density per track almost constant. The number of sectors on multiple neighboring cylinders, referred to as *zones*, is maintained at the same value to simplify bookkeeping. The number of disk zones may be higher than 20 and the number of tracks per zone may be in the thousands A consequence of zoning is that the data transfer rate from outer tracks is much higher than the inner tracks. If the outermost track has $1 + \alpha$ as many sectors as the innermost track, its transfer rate is higher by that factor, and the increase in disk capacity is roughly $1 + \alpha/2$, that is, 25% for $\alpha = 0.5$. The transfer time of frequently accessed large files can be reduced by placing them on the outermost zone (note that all tracks in a zone have the same transfer rate).

Higher recording densities have resulted in smaller diameter disks and fewer disk platters (even one), because the capacity is already met. This simplifies disk design and allows disks to rotate faster, that is, at higher RPMs, with less power and generating less heat. In fact, disk RPMs can be varied (reduced) to consume less power [Gurumurthi et al. 2003].

Higher RPMs combined with higher recording densities have resulted in higher disk transfer rates, which are of the order of tens of megabytes per second (MB/s). Higher RPMs also incur lower rotational latencies.

The disk transfer rate, which varies zone to zone, is lower than the bus data transfer rate. This mismatch is handled by the **onboard disk cache**, which buffers data that is read from disk before it is transmitted on the bus. The bus data transfer of an incompletely read block can be started as soon as enough data has been read from disk to ensure an uninterrupted transfer. This depends on the zone from which the data is being read.

Higher disk access bandwidths can be attained by accessing larger blocks, so as to amortize positioning overhead. *Track-aligned extents (traxtents)* use disk-specific knowledge to improve performance [Schindler et al. 2002]. Access time is reduced be eliminating head switching time by assigning blocks such that they do not span track or cylinder boundaries. Matching block sizes to track size and reading full tracks eliminates rotational latency, provided the disk has the zero-latency capability.

The onboard cache is also used for prefetching all or the remaining blocks of a track from which the data data was accessed. Prefetching is initiated after the disk controller detects sequentiality in the

access pattern, but disk utilization due to prefetching may be wasteful because prefetching is speculative [Patterson et al. 1995]. Caching writes and deferring their processing for improving performance might be disabled because it affects data integrity. The performance degradation introduced by prefetching can be minimized by allowing preemption. Little has been written about onboard cache management policies because they are of proprietary nature.

Most modern disk drives, such as the Maxtor Atlas 15K, come in different sizes (18, 36, and 73 GB) which is achieved by increasing the number of disks from 1 to 2 to 4 and the number of R/W heads by twice these numbers. The RPM is 15K (surprised?); hence, the rotation time is $T_{rot} = 4$ ms, there are 24 zones per surface, 61 KTPI, 32,386 cylinders, the number of 512-byte sectors per track varies from 455 to 630. The maximum effective areal density is 18 Gbits^2. The head switching time (on the same track) is less than 0.3 ms for reads and 0.48 ms for writes, while sequential cylinder switch times are 0.25 ms and 0.40 ms, respectively. The random average seek time to read (resp. write) a random block is 3.2 (resp. 3.6) ms, while the full-stroke seek is less than 8 ms. The maximum transfer rate is 74.5 MB/s, and the onboard buffer size is 8 MB. A 45-byte Reed-Solomon code [MacWilliams and Sloane 1977] is used as an error correcting code (ECC), and uncorrectable read errors occur in one per 10^{15} bits read. The access time to a small randomly placed block approximately equals the positioning time, which is $2 + 3.2 = 5.2$ ms (half of the disk rotation time added to the mean seek time), so that the maximum access rate for read requests (ignoring controller overhead) is 192.3 requests per second.

We have discussed **fixed disks** as opposed to **removable disks**. Removable disks with large form-factors were popular in the days when mini- and mainframe computers dominated the market. The removable disk was inserted into a disk drive, which resembled a washing machine with a shaft that rotated the disk. The R/W heads were attached to an arm that was retracted when the disk was being loaded/unloaded.*

A good but dated text describing magnetic disk drives is Sierra [1990]. The organization and performance of modern disk drives are discussed in Ruemmler and Wilkes [1994] and Ng [1998]. In what follows we will discuss various techniques for reducing disk access time.

21.2.2 Disk Arm Scheduling

A disk is at its best when it is sequentially transferring large blocks of data, such that it is operating close to its maximum transfer rate. Large block sizes are usually associated with **synchronous requests**, which are based on processes that generate one request at a time, after a certain "think time." Prefetching, at least "double buffering," is desirable; that is, initiate the transfer of next block while the current block is being processed. Synchronous requests commonly occur in multimedia applications, such as video-on-demand (VoD). Requests are to successive blocks of a file and must be completed at regular time intervals to allow glitch-free video viewing. Disk scheduling is usually round-based, in that a set of streams are processed periodically in a fixed order. An **admission control policy** can be used to ensure that processing of a new stream is possible with satisfactory performance for all streams and that this is done by taking into account buffer requirements [Sitaram and Dan 2000].

In contrast, **discrete requests** originate from an *infinite number of sources*. The arrival process is usually assumed to be Poisson with parameter λ, which implies (1) the arrivals in a time interval are uniformly distributed over its duration, (2) exponentially distributed interarrival times with a mean equal to $1/\lambda$, and (3) the arrival process is *memoryless*, in that the time to the next arrival is not affected by the time that has already elapsed [Kleinrock 1975]. In an OLTP system, sources correspond to concurrent transactions that generate I/O requests. A disk access is required when the requested data block is not cached at a level of the memory hierarchy preceding the disk.

It is commonly known that OLTP applications generate accesses to small, randomly placed blocks of data. For example, the analysis of the I/O trace of an OLTP workload showed that 96% of disk accesses

*IBM developed an early disk drive with 30 MB of fixed and 30 MB of removable storage, which was called Winchester in honor of its 30/30 rifle. Winchester disks are now synonymous with hard disks.

are to 4-KB and 4% to 24-KB blocks of data [Ramakrishnan et al. 1992]. The discussion in this section is therefore based on accesses to small, random blocks. This is followed by a brief discussion of sequential and mixed requests in the next section.

The default FCFS scheduling policy provides rather poor performance in the case of randomly placed data. Excluding the time spent at the disk controller, the mean service time in this case is the sum of the mean seek time, the mean latency, and the transfer time. The transfer time of small (4 or 8 KB) blocks tends to be negligibly small with respect to positioning time, which is the sum of the seek time and rotational latency.

Disk scheduling methods require the availability of multiple enqueued requests to carry out their optimization. In fact, the improvement with respect to the FCFS policy increases with the number of requests that are available for scheduling. This optimization was done by the operating system; but with the advent of the SCSI protocol, request queueing occurs at the disk.

The observation that disk queue-lengths are short was used as an argument to discourage disk scheduling studies [Lynch 1972]. Short queue-lengths can be partially attributed to **data access skew**, which was prevalent in computer installations with a large number of disks. A few disks had a high utilization, but most disks had a low utilization.

Evidence to the contrary, that queue-lengths can be significant, is given in Figure 1 in Jacobson and Wilkes [1991]. Increasing disk capacities and the volume of data stored on them should lead to higher disk access rates, based on the fact that there is an inherent **access density** associated with each megabyte of data (stale data is deleted or migrated to tertiary storage). On the other hand, larger data buffers are possible due to increased memory sizes; that is, as the disk capacity increases, so does the capacity for caching, which limits the increase in disk access rate [Gray and Shenoy 2000]. The working set of disk data in main memory was shown to be a few percent of disk capacity in one study [Ruemmler and Wilkes 1993].

Most early disk scheduling policies concentrated on reducing seek time, because of its dominant effect on disk positioning time. This is best illustrated in Jacobson and Wilkes [1991], which gives the ratio of the maximum seek times and rotation times for various disks. This ratio is quite high for some early disk drives (IBM 2314). The **shortest seek time first (SSTF)** and **SCAN** policies address this issue [Denning 1967].

SSTF serves requests from the queue according to the seek distance (i.e., with the request with the smallest seek distance and seek time served first). SSTF is a greedy policy, so that requests in some disk areas (e.g., innermost and outermost disk cylinders) will be prone to starvation if there is a heavy load at other disk areas (e.g., the middle disk cylinders).

The SCAN scheduling policy moves the disk arm in alternate directions, making stops at cylinders to serve all pending requests, so that it is expected to be less prone to starvation and to produce a smaller variance in response time than SSTF. SCAN is also referred to as the **elevator algorithm**.

Cyclical SCAN (CSCAN) returns the disk arm to one end after each scan, thus alleviating the bias in serving requests on middle disk cylinders twice. A plot of the mean response time at cylinder c (R_c) vs. the cylinder number shows that R_c is lower at the middle disk cylinders and higher at outer cylinders for scan. LOOK and CLOOK are minor variations of SCAN and CSCAN that reverse the direction of the scan as soon as there are no more requests in that direction, rather than reaching the innermost and/or outermost cylinder if there are no requests in the direction of the scan.

The **shortest access time first (SATF)** or **shortest positioning time first (SPTF)**) gives priority to requests whose processing will minimize positioning time [Jacobson and Wilkes 1991]. This policy is desirable for modern disks with improved seek times. In effect, we have a *shortest job first (SJF)* policy with the desirable property that it minimizes the mean response time among all non-preemptive policies [Kleinrock 1976]. The difference between SATF and SJF is that the service time of individual disk requests depends on the order in which they are processed.

Several simulation experiments have shown that SPTF, which minimizes positioning time, outperforms SSTF and SCAN [Worthington et al. 1994, Thomasian and Liu 2002a, Thomasian and Liu 2002b]. SPTF is a greedy policy, and appropriate precautions are required to bound the waiting time of requests (e.g., by reducing the positioning time according to waiting time).

Prioritizing the processing of one category of requests with respect to another (e.g., reads vs. writes) is a simple way to improve the performance of reads in this case. The head-of-the-line priority queueing discipline serves requests (in FCFS order) from a lower priority queue, but only when all higher priority queues are empty [Kleinrock 1976].

The SATF policy can be modified to prioritize read requests with respect to write requests as follows Thomasian and Liu [2002a] and Thomasian and Liu [2002b]. An SATF winner read request is processed unconditionally, while the service time of an SATF winner write request is compared to that of the best read request and processed only if the ratio of its service time and that of the read request is below a threshold $0 \leq t \leq 1$. In effect, $t = 1$ corresponds to "pure SATF," while $t = 0$ prioritizes reads unconditionally. There are two considerations: (1) simulation results show that doing so results in a significant reduction in throughput with respect to SATF, and (2) such a scheme should take into account the space occupied by write requests at the onboard buffer. An intermediate value of t can be selected, which improves response time while achieving a small reduction in throughput.*

SATF performance can be improved by applying **lookahead**. For example, consider an SATF winner request A, whose processing will be followed by request X, also according to the SATF policy. There might be some other request B, which when followed by Y (according to SATF) yields a total processing time $T_B + T_Y < T_A + T_X$. With n requests in the queue, the cost of the algorithm increases from $O(n)$ to $O(n^2)$. In fact, the second requests (X or Y) may not be processed at all, so that in carrying out comparisons, their processing time is multiplied by a discount factor $0 \leq \alpha \leq 1$. The improvement in performance due to lookahead is insignificant for disk requests uniformly distributed over all disk blocks, but improves performance when requests are localized.

There have been many proposals for hybrid disk scheduling methods, which are combinations of other well-known methods. We discuss two variations of SCAN and two policies that combine SCAN with SSTF and SPTF policies in order to reduce the variance of response time.

In **N-step SCAN**, the request queue is segmented into subqueues of length N and requests are served in SCAN order from each subqueue. When $N = 1$, we have the FCFS policy; otherwise, when N is large, N-step SCAN is tantamount to the SCAN policy.

FSCAN is another variation of SCAN, which has $N = 2$ queues [Coffman et al. 1972]. Requests are served from one queue, while the other queue is being filled with requests. This allows the SCAN policy to serve requests that were there at the beginning of the SCAN.

The $V(R)$ disk scheduling algorithm ranges from V(0) = SSTF to V(1) = SCAN as its two extremes [Geist and Daniel 1987]. It provides a "continuum" of algorithms combining SCAN and SSTF for $0 \leq R \leq 1$. R is used to compute the bias $d_{bias} = R \times C$, which is subtracted from the seek distance in the forward direction (C is the number of disk cylinders). More precisely, the seek distance in the direction of the scan is given as $\max(0, d_{forward} - d_{bias})$, which is compared against $d_{backward}$. The value of R is varied in simulation results to determine the value that minimizes the sum of the mean response time and a constant k times its standard deviation, which is tantamount to a percentile of response time. It is shown that for lower arrival rates, SSTF is best (i.e., $R = 0$), while at higher arrival rates ($R = 0.2$) provide the best performance.

Grouped Shortest Time First (GSTF) is another hybrid policy combining SCAN and STF (same as SATF) [Seltzer et al. 1990]. A disk is divided into groups of consecutive cylinders and the disk arm completes the processing of requests in the current group according to SPTF, before proceeding to the next group. When there is only one group, we have SPTF and with as many groups as cylinders, and we effectively have SCAN (with SPTF at each cylinder). The **weighted shortest time first (WSTF)** also considered in this study multiplies the anticipated access time by $1 - w/W_M$, where w is the waiting time and W_M

*This technique can be utilized to make the mean response of requests to a failed disk equal to the mean response times at surviving disks. This is accomplished by prioritizing disk accesses on behalf of fork-join requests with respect to others (see Section 21.5).

is the maximum waiting time (the 99th percentile of waiting time is a more meaningful metric for this purpose).

21.2.2.1 Disk Scheduling for Continuous Data Requests

Continuous data requests have an implicit deadline associated with the delivery of the next data block (e.g., video segment in a video stream) for glitch-free viewing.

The **Earliest Deadline First (EDF)** scheduling policy is a natural choice because it attempts to minimize the number of missed deadlines. On the other hand, it incurs high positioning overhead, which is manifested by a reduction in the number of video streams that can be supported. SCAN-EDF improves performance by using SCAN while servicing requests with the same deadline.

Scheduling in "rounds" is a popular scheduling paradigm, so that the successive blocks of all active requests need to be completed by the end of the round. The size of the blocks being read and the duration of the rounds should be chosen carefully to allow for glitch-free viewing. Round-robin, SCAN, or **Group Sweeping Scheduling (GSS)** [Chen et al. 1993] policies have been proposed for this purpose.

In addition to requests to continuous or C-data, media servers also serve discrete or D-data. One method to serve requests is to divide a round into subrounds, which are used to serve requests of different types. More sophisticated scheduling methods are described in [Balafoutis et al. 2003], two of which are described here.

One scheduling method serves C-requests according to SCAN and intervening D-requests according to either SPTF or OPT(N). The latter determines the optimal schedule after enumerating all $N!$ schedules ($N = 6$ is used in the chapter).

The **FAair MIxed-scan ScHeduling (FAMISH)** method ensures that all C-requests are served in the current round and that D-requests are served in FCFS order. More specifically, this method constructs the SCAN schedule for C-requests but also incorporates the D-requests in FCFS order in the proper position in the SCAN queue, up to the point where no C-request misses its deadline. D-requests can also be selected according to SPTF. The relative performance of various methods as determined by simulation studies is reported in Balafoutis et al. [2003].

21.2.3 Methods to Improve Disk Performance

These methods include (1) disk arm prepositioning, (2) disk reorganization and defragmentation, (3) log-structured file systems, and (4) active disks and free-block disk scheduling.

21.2.3.1 Disk Arm Prepositioning

When disk utilization is low, the service time of future requests can be reduced by prepositioning the disk arm. By positioning the disk arm at the middle disk cylinder, the average seek distance for random requests will be reduced from $C/3$ to $C/4$ [King 1990].

With mirrored disks it is best to position one arm at $C/4$ and another arm at $3C/4$, when all requests are reads, so that the positioning time is reduced to $C/8$ [King 1990]. When a fraction w of requests are writes, a symmetry argument can be used to place the two arms at a distance s at the opposite sides of the middle disk cylinders; that is, $1/2 \pm s$ (the maximum seek distance is set to 1). Taking the derivative of the seek distance with respect to s yields $s_{opt} = 0.25(1 - 2w)/(1 - w)$ and the expected seek distance at s_{opt} is $E[d_{min}] = 0.25 - 0.125(1 - 2w)^2/(1 - w)$. Since $s_{opt} \leq 0$ for $w \geq 1/2$, it follows that s_{opt} should be set to zero, i.e., when write requests dominate, both disk arms should be placed at the middle disk cylinders.

The disadvantage of this method is that disk arm positioning can interfere with arriving requests and the chances for this increase with the arrival rate. Preemption of disk arm positioning requests is one method to deal with this problem.

Non-work-conserving schedulers, which introduce forced idleness [Kleinrock 1976], can be used to improve performance in some cases. In a general context, this is so if a current activity is not completed and switching to another activity will result in incurring the setup time twice, once to switch to the new activity and again to return to the current activity. Setup time in processing disk requests manifests itself as positioning time. It is best that the scheduler defers the processing of a new request until it has been

ascertained that there are no further pending requests. The improvement in performance due to *anticipatory disk scheduling* is explored in the context of Apache file servers, Andrews filesystem, and the TPC-B benchmark in Iyer and Druschel [2001].

21.2.3.2 Disk Reorganization and Defragmentation

The access time to large files is improved by allocating them to contiguous disk blocks. This is not a problem if there is abundant disk space; but otherwise, a "flexible" file organization (the original UNIX file system) might allow blocks to be scattered all over the free disk space. The sequential reading of a file will then incur multiple positioning times, versus one in the best case. The Unix BSD FFS (Fast File System) achieved improved file performance by enforcing larger allocations and an improved directory organization [McKusick et al. 1984]. The performance of file systems allowing disk fragmentation can be improved by applying defragmentation programs, which even come packaged with some operating systems.

Seek distances, and hence overall performance, can be simply improved based on access frequencies, as in the well-known **organ pipe arrangement** [Wong 1980]; that is, the most popular file is placed on the middle disk cylinder and other files are placed next to it, alternating from one side to the other. File access frequencies, which vary over time, can be determined by monitoring file accesses. These frequencies are then used to adaptively reorganize the files to approximate an organ pipe organization [Akyurek and Salem 1995].

The average seek distance and time can be reduced by allocating disk files by taking into account the frequency with which files are accessed together. **Clustering algorithms** utilizing dynamic access patterns have been proposed for this purpose [Bennett and Franaszek 1977].

21.2.3.3 Log-Structured File Systems (LFS)

The **log-structured file systems (LFS)** paradigm is useful in an environment where read requests are not common, while there are many writes. This is so in a system with a large cache, where most read requests are satisfied by cache accesses and the disks are mainly used for writing (there is usually a timeout for modified data in the cache to be destaged). Caching can be at a client's workstation or PC, so that a user is involved in continuously modifying the locally cached file and saving it occasionally (or even frequently if the power supply is unreliable) at the disks of a centralized server.

LFS accumulates modified files in a large cache, which is destaged to disk in large chunks, called segments (e.g., the size of a disk cylinder), to prorate arm positioning overhead [Rosenblum and Ousterhout 1992]. Space previously allocated to altered files is designated as free space and a background garbage collection process is used to consolidate segments with free space to create almost full and absolutely empty segments for future destaging.

The write cost in LFS is a (steeply) increasing function of the utilization of disk capacity (u), which is the fraction of live data in segments (see Figure 3 in Rosenblum and Ousterhout [1992]) and a crossover point with respect to Unix's FFS occurs at $u = 0.5$. The following segment cleaning policies are investigated: (1) when should the segment cleaner be executed?, (2) how many segments should be cleaned?, (3) the choice of the segment to be cleaned, and (4) the grouping of live blocks into segments. The reader is referred to Rosenblum and Ousterhout [1992] for more details.

21.2.3.4 Active Disks and Free-Block Scheduling

Given the high value of disk arm utilization and the fact that each disk is equipped with a relatively powerful microprocessor with local memory, there have been recent proposals for **freeblock scheduling (FS)**, which utilizes opportunistic disk accesses as the arm moves in processing regular requests [Lumb et al. 2000]. FS can be used in conjunction with LFS to carry out *cleaning*, which is the defragmenting operation to prepare free segments. Other studies have considered nearest-neighbor search queries, data mining, etc. [Acharya et al. 1998, Riedel et al. 2000, Riedel et al. 2001].

The off-loading of CPU processing to the disk controller works well for simpler activities that can be localized to a disk; otherwise, activities at several disks must be coordinated at the server. In fact, despite

their slower microprocessors, the overall processing capacity at (a large number of) disks might exceed the processing capacity of the server.

Another advantage of downloading database applications to disk controllers is a reduction in the volume of data to be transmitted. This is important when the I/O bus is a potential bottleneck. For example, computing the average salary ($\bar{S}$) of N employees, whose information is held in B byte records, will require the transmission of a few bytes vs. $N \times B$ bytes, allowing more disks to be connected to the bus. In case this data resides at K disks, then the kth disk sends the local average salary $\bar{S}_k$ and the number of employees N_k to the server, which then computes $\bar{S} = \sum_{k=1}^{K} \bar{S}_k N_k / N$, where $N = \sum_{k=1}^{K} N_k$.

21.3 RAID Disk Arrays

The main motivation for **RAID (Redundant Array of Inexpensive Disks)**, as reflected by its name, was to replace *Single Large Expensive Disks (SLEDs)* used in mainframe computers with an array of small form-factor, *inexpensive*, hard disk drives utilized in early PCs [Patterson et al. 1998] (see Section 21.3.1). In Section 21.3.2, common features of RAID levels are discussed under the title of RAID Concepts. Five RAID levels — RAID1–5 — were initially introduced, and two more levels (RAID0 and RAID6) were added later [Chen et al. 1994]. We proceed to discuss all RAID levels briefly, but dedicate separate sections to RAID1 (mirrored disks) and RAID5 (rotated parity arrays). The memory hierarchy and especially the disk controller cache are discussed next. We conclude with a brief discussion or reliability modeling in RAID.

21.3.1 Motivation for RAID

A design study reported in Gibson [1992] replaces one IBM 3390 with 12 useful actuators and 10.6-inch diameter disks with 70 IBM 0661 3.5-inch disks to maintain the same capacity. The increased number of disks and their low reliability (with respect to SLEDs) led to an inexpensive system with unacceptably low reliability. This issue was addressed by introducing fault-tolerance through redundancy (the *R* in RAID). An additional 14 inexpensive disks were added to the 70 disks, seven of which were for parity and seven were spares.

An added complication associated with RAID used in conjunction with mainframe computers available from IBM (and its compatibles), which ran the MVS/390 operating system (now renamed z/OS), was that MVS issued I/O commands to variable block size (count-key-data [CKD] and extended CKD [ECKD]) disks, which were unrecognizable by *fixed block architecture (FBA)* disks, for example, the IBM 0661 Lightning disk drives, with 512-byte sectors. Two solutions are:

- Rewrite the MVS file system software and I/O routines to access FBA disks. This solution, which is only available to IBM, was perhaps too costly to undertake because possibly most of the routines were written using the low-level assembler programming language.
- Provide a RAID controller that intercepts I/O requests originating from the mainframe to ECKD disks and emulates CKD/ECKD disks on FBA disks.

In effect, the RAID controller, in addition to providing access to the disks and controlling the cache, acts as a simulator of (no longer manufactured) SLEDs on FBA disks. Controllers' front-end processors receive I/O commands addressed to cylinder, track, and block numbers (on virtual 3380 or 3390 disks), which are translated to logical block numbers. The I/O commands to SCSI or IDE drives are issued by back-end processors.

All currently manufactured disks have a small form-factor, so that the *I* in RAID now stands for *independent*. The 3.5-inch disks are the most popular form-factor for servers, desktop, and notebook PCs, while smaller form-factor disks are used in mobile systems.

New capabilities associated with z/OS (a successor to MVS and OS/390) on IBM's mainframe computers for efficient access to disk arrays [Meritt et al. 2003]. It was possible for a request to be enqueued for an

original disk drive, which had a request in progress. Given that the data on that drive was allocated over several disks made this enqueueing unnecessary when the request was for data on another physical drive. The unnecessary waiting is obviated by means of the *PAV (parallel access volume)* capability in z/OS.

21.3.2 RAID Concepts

We first introduce **striping**, which is utilized by most **RAID** levels. We then discuss fault-tolerance techniques used in RAID systems, followed by caching and prefetching in RAID systems. We finally discuss RAID reliability modeling.

21.3.2.1 Striping

Striping, later classified as RAID0 [Chen et al. 1994], is not a new concept and was used in an early, high-performance airline reservation systems. Data was stored on a few (middle) disk cylinders to reduce seek time. Striping was also used in Cray supercomputers for increased data rates, but such data rates can only be sustained by supercomputers. In this chapter we will deemphasize the topic of RAID parallelism because it is best discussed in conjunction with parallel file systems and associated applications [Jain et al. 1996, Jin et al. 2002].

Traditionally, files or datasets for commercial applications required a fraction of the capacity of a disk, so that multiple files with different access rates were usually assigned to the same disk. There was no guarantee that a random allocation of files would result in a balanced disk load and **data access skew** was observed in many systems. Several packages were developed to relocate datasets to balance the load. Local perturbations to the current allocation, rather than global reallocation, was generally considered to be sufficient [Wolf 1989].

Striping partitions (larger) files into **striping units**, which are allocated in a round-robin manner on all disks. The striping units in a row constitute a **stripe** (e.g., D0-D5, D6-D11, etc.) (see Figure 21.1).

Load balancing is attained because all disks are allocated striping units from all files, so that equal access rates to the blocks of a file result in uniform access to disks. Skew is possible for accesses to records of the same file; that is, some records can be accessed more frequently than others (e.g., according to a Zipfian distribution). The effect of highly skewed accesses to small files or nonuniform accesses to the blocks of a file is expected to be obviated by caching; that is, such small files and sets of records will tend to reside in main memory if the access rate is significant (e.g., once every "five minutes") [Gray and Reuter 1993].

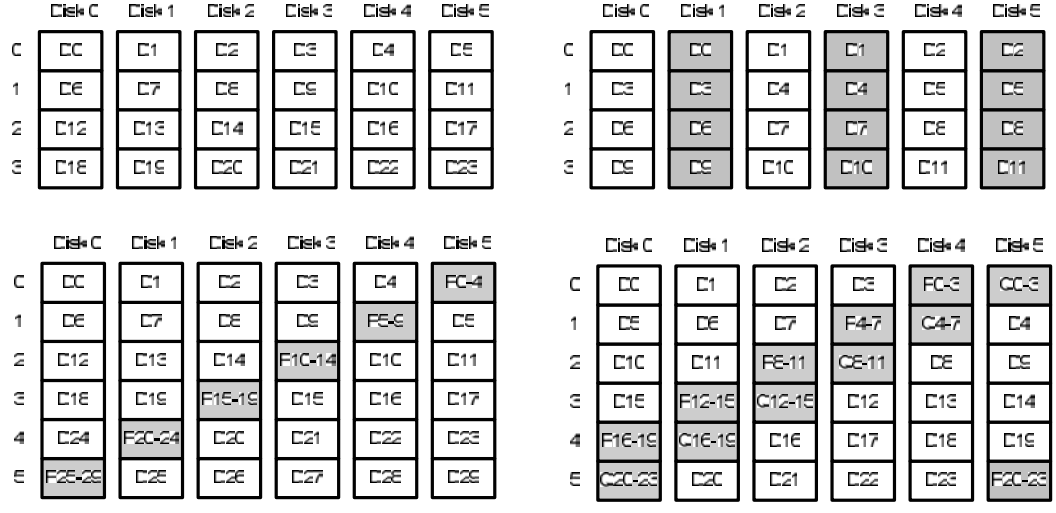

FIGURE 21.1 RAID0, RAID1, RAID5, and RAID6 layouts.

A possible disadvantage of striping is that distributing the segments of a file among several disks makes backup and recovery quite difficult. For example, if more than two disks fail in a RAID5 system, we need to reconstruct all disks, rather than just two. The **parity striping** data layout [Gray et al. 1990] described below could easily back up the data blocks at each disk separately, but then the parity blocks at all disks must be recomputed.

Striping over the disks of one array does not eliminate access skew across arrays, because each array holds a different set of files. Access skew can be eliminated by allocating data across disk array boundaries, but this requires coordination at a higher level.

The striping unit size has been an area of intense investigation. It is intuitively clear that it should be large enough to ensure that most commonly occurring request sizes can be satisfied by a single disk access. Access to small (4 or 8 KB) blocks are common in OLTP applications, while ad hoc queries and decisions support applications require accesses to much larger block sizes (e.g., 64 KB).

Data transfer time is a small fraction of arm positioning time for current disks, so that the stripe unit size should be selected to be large enough to accommodate 64-KB requests via a single disk access.

Servers run transactions at a high degree of concurrency and each transaction generates multiple I/O requests at short time intervals. Because of the high volume of I/O requests, the disks in the disk array are best utilized by not participating in parallel accesses. *Effective disk utilization* is the fraction of time that the disk is involved in "useful data transfers." Data transferred due to prefetching may not be useful because is it is not used by an application.

Accesses to very large blocks can be accommodated by large stripe units, but a very large stripe unit may introduce data access skew. In fact, very large blocks are best striped and accessed in parallel, so that the stripe unit size should be selected appropriately to optimize performance.

Parity striping maintains the regular data layout, but allows enough space on each disk for parity protection [Gray et al. 1990]. The parity blocks are placed in a separate area on each disk. Contrary to the reasoning given earlier regarding load balancing, it is argued in this paper that without striping, the user has direct control over data allocation and can ensure, for example, that two hot files are placed on two different disk drives, while this is not guaranteed in RAID5.

A performance comparison of parity striping with RAID5 is reported in Chen and Towsley [1993]. RAID5's stripe unit is assumed to be so small that there is no access skew; on the other hand, a logical request results in multiple disk accesses. The parity striped array is assumed to be susceptible to data skew, although each logical request is satisfied with one disk access. The relative performance of the two systems is compared using an analytic model based on M/G/1 queueing analysis [Kleinrock 1975].

A study for maximizing performance in striped disk arrays [Chen and Patterson 1990] estimates the size of the optimal striping unit size (KB) as the product of $S(\approx 1/4)$, the average positioning time (in milliseconds), data transfer rate (megabytes per second), and the degree of concurrency minus one, plus 0.5 KB.

Randomized data allocation schemes are a variation to striping [Salzwedel 2003]. A precomputed distribution to implement the random allocation in a multimedia storage server is presented in Santos et al. [2000] and compared with data striping.

21.3.3 RAID Fault-Tolerance and Classification

Fault-tolerance through coding techniques was proposed for early RAID designs to attain a reliability level comparable to SLEDs [Lawlor 1981, Patterson et al. 1998, Gibson 1992]. In fact, IBM's AS/400 system supported disk arrays with a check (or parity disk) [Gray et al. 1990]. Note that this capability is in addition to the error detection and correction features associated with magnetic disks. For example, the *ECC (error correcting code)* associated with each disk sector can detect long error bursts and correct several faulty bytes per sector. The failed disk in an array of disks is called an *erasure* because its location is known. A parity (resp. Hamming) code, which can normally detect single (resp. double) bit errors, can *correct* one (resp. two) bit, once the location of the erroneous bits is known.

Given that d_n denotes a single data bit on disk $1 \leq n \leq N$, then $p \leftarrow d_1 \oplus d_2 \cdots \oplus d_N$, where $\oplus$ is the *exclusive-OR (XOR)* operation. For example, given that d_1 (on disk one) is unavailable

due to disk failure, then its value can be obtained by reading all the other disks in the parity group and computing:

$$d_1 \leftarrow p \oplus d_2 \oplus \cdots d_N$$

A single check (parity) disk is utilized in RAID3 and RAID4 disk arrays. RAID3 arrays have a small striping unit (e.g., a sector) and use parallel access to expedite large data transfers. Without synchronization, the time to complete parallel I/Os will be dominated by positioning times, so that larger block sizes should be transferred to prorate the positioning time overhead.

If disks are used exclusively for parallel processing, synchronization delay for seek times can be minimized by moving the disk arms together. *Spindle synchronization*, a functionality available to SCSI disk drives, can be used to minimize synchronization time due to latency.

In RAID3, single disk failures can be handled by reading the parity disk, in addition to the surviving data disks, to reconstruct missing data on demand. Similarly, when data is being written, the bits being written are XORed to generate and write the parity. Thus efficient **full stripe writes** can be easily achieved in RAID3.

RAID4 utilizes a larger stripe unit than RAID3, so that disks are usually accessed independently. A weakness of RAID4 is that the parity disk may become a bottleneck for write-intensive applications. This effect can be alleviated by rotating the parity at the stripe unit level, which is the only difference between RAID4 and RAID5. In fact, the **left symmetric** parity organization has been shown to have several desirable properties [Gibson 1992] (see Figure 21.1).

RAID1 or mirrored disks is a degenerate case of RAID4 with one disk protected by the other (see Figure 21.1). Several schemes to combine striping with RAID1 are discussed in Section 21.4.

RAID2 disk arrays utilize a *Hamming code*, which could be used to detect the location of up to two failed disks, but because the location of these disks is already known, it can correct two disk failures [Gibson 1992]. The number of check disks K required for N data disks satisfies $2^K \geq N + K + 1$; for example, $N = 4$ requires $K = 3$. The check disk overhead decreases as N increases. Other linear coding techniques to handle multiple disk failures are described in Hellerstein et al. [1994] but these techniques have not been adopted in practice.

Two-dimensional parity protection is based on an $N \times N$ array of data disks, with N disks providing horizontal and another N disks providing vertical parity [Lawlor 1981, Gibson 1992]. More elaborate methods for two-dimensional parity are described in Newberg and Wolfe [1994]. A clever scheme that deals with the unreliability of disks, as well as other components of a RAID5 disk array, is proposed in Ng [1994].

The RAID6 classification was introduced to classify disk arrays that have two check disks and can tolerate two disk failures, such as StorageTek's Iceberg disk array [Chen et al. 1994]. Iceberg uses P + Q or more formally Reed-Solomon codes, which is a *non-binary symbol code* [MacWilliams and Sloane 1977]. Two different linear combinations of data bits, called P and Q, are computed and stored according to a left-symmetric distribution in parallel diagonals as in RAID5 (see Figure 21.1). In the case of one-disk failures, the parity P can be used to reconstruct the data, as in RAID5. In the case of two disk failures, where the data being referenced is on one of two unavailable data striping units, is the more complicated case. We need to access the corresponding $N - 2$ data blocks and the corresponding P and Q check blocks, from N out of $N + 2$ disks. To obtain the missing data block (and the one at the other failed disk), we solve two linear equations in two unknowns, except that these calculations are done over *finite fields*.

The EVENODD scheme has the same redundancy level as RAID6, which is the minimal level of redundancy, but only uses the parity operation [Blaum et al. 1995].

The previous dichotomy of order of magnitude differences in disk capacity has disappeared because SLEDs are no longer manufactured. The practice of using parity to protect against single disk failures has persisted because it has two advantages:

1. There is no disruption in data accesses. The system continues its operation by recreating requested blocks on demand, although performance is degraded.
2. The system can automatically rebuild the contents of the failed disk on a *hot spare*, if such a disk is available. A hot spare is required because some systems do not allow hot swapping (i.e., replacing a broken disk with a spare disk) while the system is running.

Self-Monitoring Analysis and Recording Technology (SMART) can be used to detect that a disk failure is imminent, so that the disk can be backed up before it fails. This is advantageous because it does not involve a complicated rebuild process, which is discussed in the context of RAID5 in Section 21.5.

21.3.4 Caching and the Memory Hierarchy

Disk blocks are cached in the database or file buffers in main memory, the cache associated with the RAID controller, and the onboard disk cache. The main advantage of the caches is in satisfying read requests, and the higher the level of the cache, the lower the latency. Data cached in main memory obviates the need for a request to the lower level of the memory hierarchy. Otherwise, the server issues an I/O request, which is intercepted by a processor at the disk array controller. An actual I/O request is issued to the disk, only if the data is not cached at this level.

Part of the disk array controller cache, which is nonvolatile storage — NVS or NVRAM — can be used to provide a *fast-write* capability; that is, there is an indication to the computer system as soon as a write to NVS is completed [Menon and Cortney 1993]. To attain a reliability comparable to disk reliability, the NVRAM might be duplexed.

Fast writes have the advantage that the destaging (writing out) of modified blocks to disk can be deferred, so that read requests that directly affect application response time can be processed at a higher priority than writes. Caching of disk blocks also occurs at the database buffer, so that when a data block is updated with a NO-FORCE transaction commit policy, only logging data has to be written to disk, while dirty blocks remain in the database buffer, until they are replaced by the cache replacement policy or by checkpointing [Ramakrishnan and Gehrke 2003].

Two additional advantages of caching in the NVS cache include: (1) dirty blocks in the cache may be modified several times, before the block is destaged to disk; and (2) multiple blocks in the cache can be processed at a much higher level of efficiency by taking advantage of disk geometry. We will discuss caching further in conjunction with RAID1 and RAID5 disk arrays.

An early study of miss ratios in disk controllers is Smith [1985]. Cache management in RAID caches is investigated via trace-driven simulations in Treiber and Menon [1995]. The destaging from the cache is started when its occupancy reaches a certain high mark and is stopped at a low mark. A more sophisticated destaging policy, which starts destaging based on the rate at which the cache is being filled, is given in Varma and Jacobson [1998].

21.3.5 RAID Reliability Modeling

Reliability and availability modeling of RAID systems was an early area of interest because the inexpensive disks were less reliable than the more expensive, large-form factor disks used in mainframes. There was also a significant increase in the number of disk drives to maintain the same capacity.

An investigation of the time-to-failure of disks showed that this time can be accurately approximated by an exponential distribution [Gibson 1992], which is a special case of the Weiball distribution [Trivedi 2002]. Denoting the parameter of the exponential distribution with λ, the **Mean Time To Failure (MTTF)** $= 1/\lambda$. To allow a simplified analysis with a Markov chain model, the repair time can be represented by an exponential distribution with rate μ, so that the **Mean Time to Repair (MTTR)** $= 1/\mu$.

Reliability modeling is concerned with determining the probability that the system will fail at a given time and the time to failure, starting with a system operating in normal mode [Trivedi 2002]. The duration of time the system has been operating in that mode is not relevant due to the *memoryless property* of the exponential distribution [Kleinrock 1975, Trivedi 2002], that the *residual lifetime* of a component with an exponential failure rate is independent of the age of the component. Given that disk failure rates and the repair rate are exponential, we can represent the behavior of the system with a Markov chain [Trivedi 2002].

A RAID5 disk array with $N + 1$ disks can be in one of three states: **normal**, **degraded**, and **failed**, which correspond to the three states of the Markov chain: $S_{N+1-i}, 0 \leq i \leq 2$, where i is the number of failed disks. At S_{N+1}, the system is in normal operating mode and the system failure rate is $(N + 1)\lambda$.

A disk failure leads to S_N, that is, operation in degraded mode. The system at S_N is repaired at rate μ, which will lead the system back to normal mode (S_{N+1}). The system at S_N fails with rate $N\lambda$, this leads to S_{N-1}, and this happens with probability $p_{fail} = N\lambda/(\mu + N\lambda)$, which is very small because μ is much larger than $N\lambda$. The transient solution to this birth-death system can be easily obtained using well-known techniques [Trivedi 2002]. The mean time to failure, which is called the **Mean Time to Data Loss (MTDL)** in this case, is the mean time to transition from S_{N+1} to S_{N-1} [Gibson 1992, Trivedi 2002]:

$$MTDL = \frac{(2N+1)\lambda + \mu}{N(N+1)\lambda^2} \approx \frac{MTTF^2}{N(N+1)MTTR}$$

More sophisticated techniques can be used to obtain the MTDL with a general repair time distribution, and it is observed that the MTDL is determined by mean repair time.

RAID6's MTDL is given in Chen et al. [1994]. Analytic solutions for the reliability and availability of various RAID models appear in Gibson [1992], which also reports experiences with an off-the-shelf reliability modeling package that was not able to solve all submitted problems. This was due to a state space explosion problem. Customized solutions that take advantage of symmetry, use state aggregation and hierarchical modeling techniques, or yield approximate solutions by ignoring highly improbable transitions are applicable. Rare event simulation is another technique to deal with this problem.

21.4 RAID1 or Mirrored Disks

This section is organized as follows. We first provide a categorization of routing and disk scheduling policies in mirrored disks. We consider two cases: when each disk has an independent queue and when the disks have a shared queue.

We next consider the scheduling of requests when an NVS cache is provided to hold write requests. This allows read requests to be processed at a higher priority than writes. Furthermore, the writes are processed in batches to take advantage of disk geometry to reduce their completion time.

Finally, we present several data layouts for mirrored disks and compare them from the viewpoint of reliability and balancedness of the load when operating with a failed disk.

21.4.1 Request Scheduling with Mirrored Disks

In addition to higher reliability, mirrored disks provide twice the access bandwidth of single disks for processing read requests, which tend to dominate write requests in many applications. This effect was first identified and quantified assuming that seeks are uniform over $(0, 1)$ in Ng [1987]. Similar results were obtained in Bitton and Gray [1988] by considering disks with C cylinders having equal capacities. Requests are unformly distributed over all disk blocks and hence disk cylinders. Sending a request to the disk providing the shorter seek distance results in a reduction in the mean seek distance from $C/3$ to $C/5$, but the mean seek distance for writes is $7C/15$, which is the expected value of the maximum of two seeks distances to update the data on both disks.

Mirrored disks can be classified into two configurations as far as their queueing structure is concerned [Thomasian et al. 2003]:

1. **Independent queues (IQ).** Read requests are immediately routed to one of the disks according to some routing policy, while write requests are sent to both disks.
2. **Shared queue (SQ).** Read requests are held in this queue, so that there are more opportunities for improving performance.

   Many variations are possible, for example, deferred forwarding of requests from the shared queue (at the router) to independent queues.

Request routing in IQ can be classified as *static* or *dynamic*. Uniform and round-robin routing are examples of static policies. Round-robin routing is simpler to implement than uniform routing and improves performance by making the arrival process more regular [Thomasian et al. 2003].

The router, in addition to checking whether a request is a read or a write, can determine other request attributes (e.g., the address of the data being accessed). Such affinity-based routing is beneficial for sequential accesses to the same file. Carrying out such requests on the same disk makes it possible to take advantage of onboard buffer hits due to prefetching.

A dynamic policy takes into account the number of requests at each disk or the composition of requests at a disk, etc. A *join the shortest queue (JSQ)* policy can be used in the first case, but this policy is known not to improve performance when requests have high variability. Simulation studies have shown that the routing policy has a negligible effect for random requests, so that performance is dominated by the local scheduling policy [Thomasian et al. 2003].

SQ provides more opportunities than IQ to improve performance because more requests are available to carry out optimization. For example, the SATF policy with SQ provides better performance than is possible with IQ, because the shared queue is twice the length of individual queues [Thomasian et al. 2003].

21.4.2 Scheduling of Write Requests with an NVS Cache

The performance of a mirrored disk system without an NVS cache can be improved by using a write-anywhere policy on one disk (to minimize disk arm utilization and susceptibility to data loss), while the data is written in place later on the primary disk. Writing in place allows efficient sequential accesses, while a special directory is required to keep track of blocks written. This is a brief description of the *distorted mirrors* method [Solworth and Orji 1991], which is one of several methods for improving mirrored disk performance.

Caching of write requests in NVS can be used to improve the performance of mirrored disks. Prioritizing the processing of read requests yields a significant improvement in response time, especially if the fraction of write requests is high. We can process write requests more efficiently by scheduling them in batches optimized with respect to the data layout on disk. The scheme proposed in Polyzois et al. [1993] runs mirrored disks in two phases; while one disk is processing read requests, the other disk is processing writes in a batch mode using CSCAN.

The performance of the above method can be improved as follows Thomasian and Liu [2003]: (1) eliminating the forced idleness in processing write requests individually; (2) using SATF or preferably an exhaustive enumeration, which is only possible for sufficiently small batch sizes, say 10, instead of CSCAN, to find an optimal destaging order; (3) introducing a threshold for the number of read requests, which when exceeded, defers the processing of write batches.

21.4.3 Mirrored Disk Layouts

With **mirrored pairs**, which is the configuration used in Tandem's NonSTOP SQL, the read load on one disk is doubled when the other disk fails.

The **interleaved declustering** method used in the Teradata/NCR DBC/1012 database machine (1) organizes disks as clusters of N disks, and (2) designates one half of each disk as the primary data area and the second half as secondary, which mirrors the primary data area. The primary area of each disk is partitioned conceptually into $N-1$ partitions, which are allocated in round-robin manner over the secondary areas of the other $N-1$ disks. If one disk fails, the read load on surviving disks will increase only by a factor of $1/(N-1)$, as opposed to the doubling of load in standard mirrored disks. This method is less reliable than mirrored pairs in that the failure of any two disks in a cluster will result in data loss, while with mirrored pairs data loss occurs if two disks constituting a pair fail, which is less likely.

The **group rotate declustering** method is similar to interleaved declustering, but it adds striping. Copy 1 is a striped array and copy 2 stores the stripes of copy 1 in rotated manner [Chen and Towsley 1996]. Let's assume copy 1 and 2 both have four disks. The stripe units in the second row of copy 1 ($F5$, $F6$, $F7$, $F8$)

will appear as ($f8, f5, f6, f7$) in copy 2, and this rotation continues. There is no rotation when (row_number) mod(4) = 0. This data layout has the advantage that the load of a failed disk is uniformly distributed over all disks, but has the drawback that more than two disk failures are more likely to result in data loss than standard mirrored disks or the following method.

The **chained declustering** method alleviates the aforementioned reduced reliability problem [Hsiao and DeWitt 1993]. All N disks constitute one cluster. Each disk has a primary and secondary area, so that the primary data on each disk is allocated on the secondary area of the following disk modulo(N). Consider four disks denoted by $D0$ through $D3$ whose contents are given as $[P0, S3]$, $[P1, S0]$, $[P2, S1]$, $[P3, S2]$, where P and S denote the primary and secondary data, respectively. Assume that originally all read requests are sent to the primary data and have an intensity of one request per time unit. When D1 is broken, requests can be routed to sustain the original load, while keeping disk loads balanced, as follows: $[P0, (1/3)S3]$, $[(1/3)P2, S1]$, $[(2/3)P3, (2/3)S2]$.

21.5 RAID5 Disk Arrays

In this section we describe the operation of a RAID5 system in normal, degraded, and rebuild modes. We also describe some interesting variations of RAID5.

21.5.1 RAID5 Operation in Normal Mode

Reads are not affected, but updating a single data block d_{old} to d_{new} requires the updating of the corresponding parity block p_{old}: $p_{new} \leftarrow d_{old} \oplus d_{new} \oplus p_{old}$, When d_{old} and p_{old} are not cached, the writing of a single block requires four disk accesses, which is referred to as the **small write penalty**. This penalty can be reduced by carrying out the reading and writing of data and parity blocks as **read-modify-write (RMW)** accesses, so that extra seeks are eliminated. On the other hand, it may be possible to process other requests opportunistically, before it is time to write after a disk rotation.

Simultaneously starting the RMW access for data and parity blocks may result in the parity disk being ready for writing *before* the data block has been read. One way to deal with this problem is to incur additional rotations at the parity disk until the necessary data becomes available. A more efficient way is to start the RMW for the parity block only after the data block has been read. Such precedence relationships are best represented by dags (directed acyclic graphs) [Courtright 1997].

Several techniques have been proposed to reduce the write overhead in RAID5. One method provides extra space for parity blocks, so that their writing incurs less rotational latency. These techniques are reviewed in Stodolski et al. [1994], which proposes another technique based on batch updating of parity blocks. The system logs the "difference blocks" (exclusive-OR of the old and new data blocks) on a dedicated disk. The blocks are then sorted in batches, according to their disk locations, so as to reduce the cost of updating the parity blocks.

Modified data blocks can be first written onto a duplexed NVS cache that has the same reliability level as disks. Destaging of modified blocks due to write requests and the updating of associated parity blocks can therefore be deferred and carried out at a lower priority than reads.

The small write penalty can be eliminated if we only have full stripe writes, so that the parity can be computed on-the-fly. Such writes are rare and can only occur when a batch application is updating all of the data blocks in the dataset sequentially. On the other hand, the aforementioned LFS paradigm can be used to store a stripe's worth of data in the cache to make such writes possible. The previous version of updated blocks is designated as free space, and garbage collection is carried out by a background process to convert two half-empty stripes to one full and one empty stripe. The **log-structured array (LSA)** proposed and analyzed in Menon [1995] accomplishes just that. While this analysis shows that LSA outperforms RAID5, it seems that updates of individual blocks will result in a data allocation that does not lend itself to efficient sequential data access, unless, of course, the block size is quite large, say 64 KB, so as to accommodate larger units of transfer for database applications.

21.5.2 RAID5 Operation in Degraded Mode

A RAID5 system can continue its operation in *degraded mode* with one failed disk. Blocks on the failed disks can be reconstructed on demand, by accessing all the corresponding blocks on surviving disks according to a *fork-join request* and XORing these blocks to recreate the missing block. A fork-join request takes more time, which is the maximum of the times at all disks.

Given that we have a balanced load due to striping and that each one of the surviving disks must process its own requests, in addition to the fork-join requests, results in a doubling of disk loads when all requests are reads.

In the case of write requests, there is no need to compute the parity block if the disk on which the parity resides is broken. In case d_{old} is not cached and the corresponding data disk, say the $N+1$st disk, is broken, then the parity block that resides on disk one can be computed as: $p_1 \leftarrow d_2 \oplus d_3 \ldots d_N$.

Given the increase in RAID5 disk utilizations in degraded mode, the disk utilizations should be below 50% in normal mode, when all requests are reads, although higher initial disk utilizations are possible otherwise. **Clustered RAID**, proposed in Muntz and Lui [1990], solves this problem using a *parity group size* that is smaller than the number of disks. This means that only a fraction of disks are involved in rebuilding a block [Muntz and Lui 1990], so that less intrusive reconstruction is possible. Complete or full block designs require infinite capacity, so that **Balanced Incomplete Block Designs (BIBDs)** [Hall 1986] have been proposed to deal with finite disk capacities [Ng and Mattson 1994, Holland et al. 1994]. **Nearly random permutations** is a different approach [Merchant and Yu 1996].

Six properties for ideal layouts are given in Holland et al. [1994]:

1. Single failure correcting: the stripe units of the same stripe are mapped to different disks.
2. Balanced load due to parity: all disks have the same number of parity stripes mapped onto them.
3. Balanced load in failed mode: the reconstruction workload should be balanced across all disks.
4. Large write optimization: each stripe should contain $N-1$ contiguous stripe units, where N is the parity group size.
5. Maximal read parallelism: reading $n \leq N$ disk blocks entails accessing different n disks.
6. Efficient mapping: the function that maps physical to logical addresses is easily computable.

The *Permutation Development Data Layout (PDDL)* is a mapping function described in Schwartz [1999] that has excellent properties and good performance in light loads (like the PRIME data layout [Alvarez et al. 1998]) and heavy loads (like the DATUM data layout [Alvarez et al. 1997]).

21.5.3 RAID5 Operation in Rebuild Mode

In addition to performance degradation, a RAID5 system operating in degraded mode is vulnerable to data loss if a second disk failure occurs. If a hot spare is provided, the rebuild process should be initiated as soon as possible when a disk fails. This is important because a failed disk may not be replaceable without bringing the system down (i.e., disrupting its operation).

Instead of wasting the bandwidth of the hot spare, the **distributed sparing** scheme distributes the spare areas over $N+2$ disks [Thomasian and Menon 1997]. Distributed sparing is also used in PDDL [Schwartz 1999]. The drawback of hot sparing is that a second round of data movement is required to return the system to its original configuration.

The **parity sparing** method utilizes two parity groups, one with $N+1$ and the other with $N'+1$ disks, which are combined to form a single group with $N+N'+1$ disks after one disk fails [Chandy and Reddy 1993].

Two variations of RAID5 rebuild have been proposed in the research literature. The **disk-oriented rebuild** scheme reads successive disk tracks from surviving disks, when the disk is idle. These tracks are XORed to obtain the track on the failed disk, which is then written onto a spare disk (or a spare area in distributed sparing). The performance of this method is studied via simulation in Holland et al. [1994] and analytically in Thomasian and Menon [1997]. The **stripe-oriented rebuild** scheme synchronizes at the stripe level, or more generally at the level of the *rebuild unit*. Its performance is evaluated via simulation and

shown to be inferior compared to the former rebuild method in Holland et al. [1994]. Another technique is to allow one rebuild request at a time, which is evaluated in Merchant and Yu [1996] as a *queueing system with permanent customers* [Boxma and Cohen 1991].

A hybrid RAID1/5 disk array called AutoRAID is described in Wilkes et al. [1996], which provides RAID1 at the higher level (of disk) storage hierarchy and RAID5 at the lower level. Data is automatically transferred between the two levels to optimize performance.

21.6 Performance Evaluation Studies

In this section we provide an overview of analytic and simulation studies to evaluate the performance of disks and disk arrays. Attention is also paid to simulation tools available for this purpose.

A recent review article on performance analysis of storage systems is Shriver et al. [2000], which also pays attention to tertiary storage systems.

21.6.1 Single Disk Performance

Disk simulation studies can be classified into trace- and random-number-driven simulations. Trace-driven simulation is based on traces of I/O activity. A simplified entry in the trace consists of a timestamp, disk block (sector) numbers, number of blocks being accessed, and whether a request is a read or a write. The trace is run through the simulator of a particular disk drive to determine the performance characteristic of interest (e.g., the mean response time of disk accesses).

An influential study based on a trace-driven simulation showed that an extremely detailed model is required to achieve high accuracy in estimating disk performance [Ruemmler and Wilkes 1994]. A complementary simulation study that uses trace- and random-number-driven simulations is Worthington et al. [1994].

One of the problems with simulating disk drives is having access to their detailed characteristics. Although such information is usually published in a manual, finding this information is a time-consuming task. More importantly, some important information might be missing. The DIXTRAC tool automates the extraction of disk parameters [Schindler and Ganger 1999], which can be used by the DiskSim simulation tool [DiskSim]. Extracted disk parameters are available at [Diskspecs]. DiskSim is a successor to RAIDframe, which can be used for simulation of disk arrays, as well as fast prototyping [Courtright et al. 1996].

An I/O trace, in addition to being used in trace-driven simulations, can be analyzed to characterize the I/O request pattern, which can then be used in a random-number-driven simulation study. A positive aspect of the latter approach is that it is easy to vary the arrival rate of requests and especially the characteristics of disk requests, while basing these variations on a realistic workload.

Requests to a modern disk drive are specified by the address of the first (logical) block (sector) number and the number of blocks to be transferred. Logical block numbers are translated into physical block numbers by the disk controller, taking into account spare areas, as well as faulty sectors and tracks. It is a straightforward task to use the trace obtained on one disk model to estimate performance on another disk model, but there are some complications, such as track alignment, in case the first data allocation was optimized for the particular disk drive.

The analysis of an I/O trace for an OLTP environment showed that 96% of requests are to 4-KB blocks and 4% to 24-KB blocks [Ramakrishnan et al. 1992]. Other notable disk I/O characterization studies are Ruemmler and Wilkes [1992], Hsu et al. [2001], and Hsu and Smith [2003], where the second study is concerned with I/O requests at the logical level.

Referring back to the distinction between discrete and synchronous requests, it is meaningful to vary the arrival rate of discrete requests, although this may not be so for synchronous requests since a synchronous request is generated only after a previous one is completed.

In random-number-driven or analytic studies, the workload at a single disk is characterized by the arrival process, request sizes, and the fraction of requests of a certain size, the fraction of reads and writes, and the spatial distribution of requests (e.g., random or sequential).

The sum of the average positioning time with an FCFS policy and data transfer time yields the mean service of requests and hence the maximum throughput of the disk (λ_{max}). The disk controller overhead is known, but may be (partially) overlapped with disk accesses, so we will ignore it in this discussion. This is a lower bound to λ_{max} because positioning time can be reduced by judicious disk scheduling and we have ignored the possibility of hits at the onboard disk cache. Note that the hit ratio at the onboard cache is expected to be negligibly small for random I/O requests.

In dealing with discrete requests, most analytic and even simulation studies assume Poisson arrivals. With the further assumption that disk service times are independent, the mean disk response time with FCFS scheduling can be obtained using the *M/G/1 queueing model* [Kleinrock 1975, Trivedi 2002]. There is a dependence among successive disk requests as far as seek times are concerned, because an access to a middle disk cylinder is followed by a seek of distance $C/4$ on the average for random requests (C is the number of disk cylinders), while the distance of a random seek after an access to the innermost or outermost disk cylinders is $C/2$ on the average. Simulation studies have shown that this queueing model yields fairly accurate results.

The analysis is quite complicated for other scheduling policies [Coffman and Hofri 1990], even when simplifying assumptions are introduced to make the analysis tractable. For example, the analysis of the SCAN policy in Coffman and Hofri [1982] assumes: (1) Poisson arrivals to each cylinder; (2) the disk arm seeks cylinder-to-cylinder, even visiting cylinders not holding any requests; and (3) the processing at all cylinders takes the same amount of time. Clearly, this analysis cannot be used to predict the performance of the SCAN policy in a realistic environment.

Most early analytic and simulation studies were concerned with the relative performance of various disk scheduling methods; for example, SSTF has a better performance than FCFS, at high arrival rates, and SCAN outperforms SSTF [Coffman and Hofri 1990]. Other studies propose a new scheduling policy and carry out a simulation study to evaluate its performance with respect to standard policies [Geist and Daniel 1987, Seltzer et al. 1990]. Simulation studies of disk scheduling methods, which also review previous work, are Worthington et al. [1994], Thomasian and Liu [2002a], and Thomasian and Liu [2002b].

There have been numerous studies of multidisk configurations, where the delays associated with I/O bus contention are taken into account. **Rotational Positing Sensing (RPS)** is a technique to detect collisions when two or more disks connected to a single bus are ready to transmit at the same time, in which case only one disk is the winner and additional disk rotations are incurred at the other disks, which can result in a significant increase in disk response time. There is also a delay in initiating requests at a disk if the bus is busy. Reconnect delays are obviated by onboard caches, but a detailed analysis requires a thorough knowledge of the bus protocols Approximate analytic techniques to analyze the disk subsystem of mainframe computer systems given in Lazowska et al. [1984] are *not* applicable to current disk subsystems.

Bottleneck analysis is used in Section 7.11 in Hennessey and Patterson [2003] to determine the number of disks that can be supported on an I/O bus and an M/M/1 queueing analysis to estimate mean response time [Kleinrock 1975].

21.6.2 RAID Performance

There have been very few performance (measurement) studies of commercial RAID systems reported in the open literature. There have been several RAID prototypes — Hager at IBM, RAID prototypes at Berkeley [Chen et al. 1994], the TickerTAIP RAID system at HP [Cao et al. 1994], the Scotch prototype at CMU [Gibson et al. 1995], AutoRAID at HP Labs [Wilkes et al. 1996] — and performance results have been reported for most of them. Such studies are important in that they constitute the only way to identify bottlenecks in real systems and to develop detailed algorithms to ensure correct operation [Courtright 1997]. Some prototypes are developed for the purpose of estimating performance, in which case difficult-to-implement aspects, such as recovery, are usually omitted [Hartman and Ousterhout 1995].

One reason for the lack of performance results for RAID may be due to the unavailability of a common benchmark, but steps in this direction have been taken recently [Storage Performance Council 2003].

A performance evaluation tool for I/O systems is proposed in Chen and Patterson [1994] that is self-scaling and adjusts dynamically to the performance characteristics of the system being measured.

There have been numerous analytical and simulation studies of disk arrays. Markovian models (with Poisson arrivals and exponential service times) have been used successfully to investigate the relative performance of variations of RAID5 disk arrays [Menon 1994]. Several performance evaluation studies of RAID5 systems have been carried out based on an M/G/1 model. Most notably, RAID5 performance is compared with parity striping [Gray et al. 1990] in Chen and Towsley [1993] and with mirrored disks in Chen and Towsley [1996]. An analysis of clustered RAID in all three operating modes appears in Merchant and Yu [1996]. An analysis of RAID5 disk arrays in normal, degraded, and rebuild mode appears in Thomasian and Menon [1994] and Thomasian and Menon [1997], where the former (resp. latter) study deals with RAID5 systems with dedicated (resp. distributed) sparing. These analyses are presented in a unified manner in Thomasian [1998], which also reviews other analytical studies of disk arrays.

An analytic throughput model is reported in Uysal et al. [2001], which includes a cache model, a controller model, and a disk model. Validation against a state-of-the-art disk array shows an average 15% prediction error.

A simulation study of clustered RAID is reported in Holland et al. [1994], which compares the effect of disk-oriented and stripe-oriented rebuild. The effect of parity sparing on performance is investigated in Chandy and Reddy [1993]. The performance of a RAID system tolerating two-disk failure is investigated in Alvarez et al. [1997] via simulation, while an M/G/1-based model is used to evaluate and compare the performance of RAID0, RAID5, RAID6, RM2, and EVENODD organizations in Han and Thomasian [2003].

It is difficult to evaluate the performance of disk arrays via a trace-driven simulation because, in effect, we have a very large logical disk whose capacity equals the sum of the capacities of several disks. A simulation study investigating the performance of a heterogeneous RAID1 system (MEMS-based storage backed up by a magnetic disk) is reported in Uysal et al. [2003]. Specialized simulation and analytic tools have been developed to evaluate the performance of MEMS-based storage; see, for example, Griffin et al. [2000].

A comprehensive trace-driven simulation study of a cached RAID5 design is reported in Treiber and Menon [1995], while Varma and Jacobson [1998] consider an improved destaging policy based on the rate at which the cache is filled as well as its utilization.

21.6.3 Analysis of RAID5 Systems

As discussed, a tractable analysis of RAID5 systems can be obtained by postulating a Poisson arrival process. Two complications are estimating the mean response time of fork-join requests and the rebuild time. Analytical results pertinent to these analyses are presented here.

21.6.3.1 Fork-Join Synchronization

Analytic solutions of fork-join synchronization are required for the analysis of disk arrays in degraded mode (e.g., to estimate the mean response time of a read request to a failed disk). This results in read requests to all surviving disks, so that we need to determine the maximum of response times. In what follows, we give a brief overview of useful analytical results.

In a 2-way fork-join system with two servers, Poisson arrivals (with arrival rate λ), exponential service times (with rate μ), it has been shown that $R_2^{F/J} = (1.5 - \rho/8)R$, where ρ is the utilization factor $\rho = \lambda/\mu$ and $R = 1/(\mu - \lambda)$ is the mean response time of an M/M/1 queueing system.

Curve-fitting to simulation results was used in Nelson and Tantawi [1988] to extend this formula to $K > 2$ servers:

$$R_k^{F/J} = \left[\alpha(\rho) + (1 - \alpha(r))\frac{H_k}{H_2}\right] R_2^{F/J}$$

where $\alpha(\rho) = 4\rho/11$ and $H_K = \sum_{k=1}^{K} 1/k$.

The assumption that arrivals are Poisson and that service times are exponential is relaxed in Varma and Makowski [1994], which combines heavy and light traffic limits to derive an interpolation approximation. Validation against simulation shows this approximation to be quite accurate.

The above fork-join synchronization delay formula was utilized in several analyses of RAID5 disk arrays with Markovian assumptions: Poisson arrivals and exponential service times (see, e.g., Menon [1994]). A more careful examination of RAID5 operation in degraded mode shows that each surviving disk, in addition to processing fork-join requests, also processes its regular workload, so that these requests interfere with fork-join requests. It is stated in Nelson and Tantawi [1988] that the above formula also holds when there is interfering traffic, but, in fact, $R_K^{F/J}$ is a lower bound and R_K^{max} (the expected value of the maximum of response times at individual nodes) tends to be an upper bound [Thomasian 1997]. This is because interfering requests make the components of a fork-join request less dependent. The two bounds are quite close when the service time has a small coefficient of variation, but interpolation is required otherwise.

Computing the expected value of the maximum is trivial when the response time distributions are exponentially distributed, which is the case with exponential arrivals (with rate λ) and service times (with rate μ). For an N-way fork-join queueing system with mean response time R at one of the servers we have: $R_N^{max} = R\sum_{n=1}^{N} 1/n$, where $R = 1/(\mu - \lambda)$ [Trivedi 2002]. Note that $R_2^{max} = 1.5R$, so that as $\rho \to 1$ $R_2^{max} - R_2^{F/J} \to R/8$.

The coefficient of variation of response time is $c_R = \sigma_R/R$, where σ_R is the standard deviation of response time, and tends to be smaller than one when requests are to small uniformly distributed disk blocks. In this case, the response time distribution can be approximated by an Erlang distribution [Kleinrock 1975, Trivedi 2002]. This distribution consists of k exponential stages, each with a rate $k\mu$, so that the mean is $\bar{x} = 1/\mu$ and the *coefficient of variation* $c_R^2 = 1/K$. Given R and c_R^2, the parameters of the Erlang distribution are $\mu = 1/R$ and $k = ceil(1/c_R^2)$. More generally, for any c_R^2, the response time can be approximated by a Coxian distribution [Kleinrock 1975, Trivedi 2002], but there are more parameters to be estimated in this case [Chen and Towsley 1993]. Given the mean and standard deviation of disk response time, R_N^{max} can be obtained approximately using the *extreme value distribution* $R_N^{max} = R + \sqrt{6}\sigma_R ln(N)/\pi$. More details are given in Thomasian [1997] and Thomasian [1998].

21.6.3.2 Vacationing Server Model for Rebuild

The vacationing server model for M/G/1 queueing systems is used to analyze disk-oriented rebuild in a RAID5 system with dedicated sparing in Thomasian and Menon [1994]. This analysis is extended to distributed sparing in Thomasian and Menon [1997] by introducing an additional iterative step that makes the analysis more complicated. Only the former analysis is described here for the sake of brevity.

Each disk behaves as an M/G/1 queue that undergoes busy and idle periods. After the disk has been idle, a busy period starts when a new request arrives. After completing the processing of (external) requests and the end of the *busy period* [Kleinrock 1975], a sequence of tracks are read by the rebuild process and each one of these reads is considered a *vacation* [Kleinrock 1975]. The durations of the vacations are different because the first track being read (in an idle period) incurs a seek, while the succeeding track reads do not incur a seek. This behavior can be modeled as a vacationing server model where successive vacations have different distribution times [Takagi 1991]. No new vacations are started after the arrival of the first external request.

The service time of the first request is elongated by the residual vacation time (with mean $\bar{r}$) [Thomasian and Menon 1994]. The busy period started by this first unusual request is different from an ordinary busy period and is called a *delay cycle* [Kleinrock 1976]. The mean waiting time for a regular disk request is $\bar{r} + W$, where W is given by the Pollaczek-Khinchine formula [Kleinrock 1975]; in other words, $\bar{r}$ is added to the mean waiting time of all requests, rather than just the first one.

The mean cycle time, defined as the sum of a delay cycle and the vacation time, can be easily computed. The mean rebuild time is equal to the number of tracks divided by the mean number of tracks read per cycle time. There is the additional complication that the rate of rebuild speeds up as more and more read requests are carried out via *read redirection*, by reading rebuilt data directly from the spare disk [Muntz and Lui 1990]. Zoning is a similar complication, and can be dealt with similarly by discretizing the solution space and solving a few cylinders at a time.

21.7 Data Allocation and Storage Management in Storage Networks

Two confusing new terms are **Storage Area Networks (SAN)** and **Network Attached Storage (NAS)** (as opposed to **Direct Attached Storage (DAS)**). NAS embeds the file system into storage, while a SAN does not [Gibson and Van Meter 2002]. A SAN is typically based on Ethernet while NAS is based on Fibre Channel (note spelling). Four typical NAS systems are described in Gibson and Van Meter [2002]:

1. Storage appliances such as products from Network Appliance and SNAP!
2. NASD (network attached secure devices), a project at CMU's Parallel Data Laboratory [Gibson et al. 1998]
3. The Petal project whose goal was to provide easily expandable storage with a block interface [Lee and Thekkath 1996]
4. **The Internet SCSI (iSCSI)** protocol which implements the *SCSI (Small Computer System Interface)* protocol on top of TCP/IP. Two recent publications on this topic are Sarkar et al. [2003] and Meth and Satran [2003].

This section is organized into two subsections. We first review the requirements of a storage network, and then proceed to review recent work at HP Labs in the area of storage allocation.

21.7.1 Requirements of a Storage Network

A storage network is defined as a set of disk drives connected by a network that can be a simple SCSI bus, an Ethernet switch, or a peer-to-peer network, where the disk drives are associated with cooperating servers. Requirements for a storage network can be summarized as follows [Salzwedel 2003]:

1. **Space and access balance.** The requirement to store additional data is growing rapidly. Given that disk capacity is cheap, the storage capacity of the network can be easily expanded. There is a need to allocate data carefully to utilize disk capacities and balance disk utilizations. We have already discussed striping and randomized methods for this purpose in our discussion of RAID5 systems.

   This problem is related to the *File Assignment Problem (FAP)* [Dowdy and Foster 1982]. Given the characteristics of the disk system and the access rates of files, FAP searches for a nearly optimal assignment of files to disks to optimize some performance measure such as disk response times [Wolf 1989]. Without such an optimized allocation, disks are susceptible to access skew. Recent work in this area is discussed in the following subsection.
2. **Availability.** High data availability can be attained utilizing various forms of redundancy, for example, mirroring, as well as parity. A distributed RAID system is described in Hartman and Ousterhout [1995]. It uses striping (to balance the load) and parity, but in fact it is a log-structured array (LSA) [Menon 1995], in that data from various clients is combined into a single stream, which is written as one stripe.
3. **Resource efficiency.** Higher availability implies wasted disk capacity in the form of check disks and even replicated data. It is important to quantify this overhead against higher performance and availability.
4. **Access efficiency.** This is determined by time and space requirements to access the data.
5. **Heterogeneity.** In a smaller computer installation, it is possible to upload disk data onto tape, replace old disks with new disks, and recreate the data on the new disks. Index building for relational tables may require a considerable amount of processing time, however, especially when there are several indices per table. With the advent of higher-capacity disks and the need for high availability, this is not possible in a large distributed system. The introduction of new disks, heterogeneous or not, should introduce as little disruption as possible. Furthermore, it might not be economically viable to replace all old disks with new disks, just for the sake of maintaining homogeneity.

There have been several studies of data layout with heterogeneous disks, and these are reviewed in Section 5 of Salzwedel [2003]. Consider the data layout for RAID0 in a heterogeneous disk array with N disks, N' of which are large. The stripe width can be set to N up to the point where the smaller disks are filled, and the striping is continued with a stripe width $N - N'$ thereafter. The data layout becomes more complicated with RAID5 because we will have variable stripe sizes and there is a need to balance disk loadings for updating parity. Solutions to these and other problems are given in Cortes and Labarta [2001].

6. **Adaptivity.** This is a measure of how fast can the system adapt to increases in the data volume and the addition of more disk capacity. More specifically, a fraction of the existing data must be redistributed, and the efficiency of this process is called *adaptivity*. The *faithfulness* property is to balance disk capacity utilizations, so that given m data units and that the ith disk has a fraction d_i of the total capacity, then this disk's share of the data is $(d_i + \epsilon)m$, where ϵ is a small value [Salzwedel 2003]. The issue of speed (access bandwidth) should also be taken into consideration because it is an important attribute of disk drives.

   Adaptivity can be measured by *competitive analysis*, which measures the efficiency of a scheme as multiples of an optimal scheme, which ensures faithfulness. For example, a placement scheme will be called c-competitive if for any sequence of changes it requires the replacement of, at most, c times the number of data units needed by an "optimal adaptive and perfectly faithful strategy" [Salzwedel 2003].
7. **Locality.** This is a measure of the degree of communication required for accessing data.

21.7.2 Storage Management

There is renewed interest in this area because of user demand for more storage and predictable performance. The former issue is easy to deal with because storage costs are dropping; however, the latter issue is a difficult problem because there are numerous schemes for storage structures with different performance characteristics, yet limited knowledge is available about the workload.

We briefly describe recent studies at HP Labs in this direction [E. Anderson et al. 2002]. Hyppodrome is conceptually an "iterative storage management loop" consisting of three steps: (1) design new system, (2) implement design, (3) analyze workload and go back to (1). The last step involves I/O tracing and its analysis to generate a workload summary, which includes *request rates and counts*; *run counts*, that is, mean number of accesses to contiguous locations, (device) *queue-length*, *on and off times* (mean period a stream is generating data); and *overlap fraction* of these periods. The *solver* module takes the workload description and outputs the design of the system that meets performance requirements.

Hyppodrome uses Ergastulum to generate a storage system design [E. Anderson et al. 2003]. The inputs to Ergastulum are a workload description, a set of potential devices, and user-specified constraints and goals [Wilkes 2001]. The workload description is in the form of *stores* (static aspect of datasets, such as size) and *streams* (dynamic aspects of workload: access rates, sequentiality, locality, etc.). There is an intermediate step for selecting RAID levels, configuring devices, and assigning stores onto devices. An *integer linear programming* formulation is easy, but it is very expensive to solve (takes several weeks to run) for larger problems. Ergastulum uses *generalized best-fit bin packing* with *randomization* to solve the problem quickly, thus benefiting from an earlier tool, Minerva, which was a solver for *attribute managed storage* [Alvarez et al. 2001].

21.8 Conclusions and Recent Developments

We have given a lengthy review of secondary storage systems, which are materialized as magnetic disks and disk arrays. We have emphasized methods that can be used to improve performance, especially in the form of disk scheduling.

While there are many varieties of disk arrays, only two — mirrored disks (RAID1) and rotated parity arrays (RAID5) — are popular. There are also commercial disk arrays with P + Q coding, one of which is

also a log-structured array, and the AutoRaid dynamically adaptive RAID system. Much more work remains to be done to gain a better understanding of the performance of these systems. We have therefore covered analytic and simulation methods that can be used to evaluate the performance of disks and disk arrays.

While magnetic disks serve as a component, there is a significant amount of computing power associated with each disk that can be utilized for many useful functions. While there are many remaining problems to be solved, this is similar to a step taken by the Intelligent RAM (IRAM) project, which enhanced RAMs with processing power (or vice versa) [Kozyrakis and Patterson 2003].

The Storage Networking Industry Association (www.snia.org), among its other activities, has formed the Object-based Storage Devices (OSD) work group, which is concerned with the creation of self-managed, heterogeneous, shared storage. This entails moving low-level storage functions to the device itself and providing the appropriate interface. A recent paper on this topic is Mesnier et al. [2003].

The InfiniBand™ Architecture (IBA) was started because "processing power is substantially outstripping the capabilities of industry-standard I/O systems using busses" [Pfister 2002]. Commands and data are transferred among hosts as messages (not memory operations) over point-to-point connections.

Finally, there is increasing interest in constructing *information storage systems* that provide availability, confidentiality, and integrity policies against failures and even malicious attacks; see, for example, Wylie et al. [2000].

Acknowledgments

We acknowledge the support of NSF through Grant 0105485 in Computer System Architecture.

Defining Terms

Error correction code (ECC): ECC is computed for each sector and used to detect and even correct (within some limits) storage errors.

Floppy disks: A flexible plastic disk coated with magnetic material and enclosed in a protective envelope.

Gigabyte (GB): 2^{30} or approximately one thousand million bytes. Similarly, kilobyte (KB) is 2^{10} bytes, megabyte (MB) is 2^{20} or approximately one million bytes, and petabyte is $2^{40} \approx 10^{12}$ bytes. Powers of ten are used for disk sizes and powers of two for main memory sizes.

Hard disk drive (HDD): Consists of one or more rigid disk platters that are coated with magnetic material on which data is recorded. Also called Winchester disk drives for historical reasons.

Integrated device electronics (IDE): Disk drive interface, which includes drive controller and electronics. It is related to ATA (AT attachment) interface, which originated with IBM's PC AT (advanced technology) in mid-1980s.

Millisecond: One thousandth of a second. Similarly, a microsecond is one millionth (10^{-6}) of a second, and a nanosecond is 10^{-9} seconds.

Mirrored or Shadowed Disk: The same datasets are replicated on two disks. This corresponds to RAID level 1 or RAID1.

MTBF, MTTR, MTDL: Mean Time Between Failures, Mean Time To Repair, Mean Time to Data Loss.

Network Attached Storage (NAS): Storage, such as a disk array, that is directly attached to a network (see SAN) rather than a server.

Network File System (NFS): A file system originating from Sun Microsystems, but now accepted as an IETF (Internet Engineering Task Force [IETF]) standard for file services on TCP/IP networks.

Offline/nearline/online storage: Infrequently used data is stored in offline storage, which usually consists of a tape archive. Nearline storage refers to automated type library, while online storage for frequently accessed data usually resides on hard disk drives (as opposed to floppy disks).

Parity blocks: Blocks computed bit-by-bit using the exclusive-OR (XOR) operation from data blocks. It is used in RAID3, RAID4, and RAID5 to reconstruct data on a failed disk.

RAID (Redundant Array of Inexpensive Disks): An array of disks organized to provide higher availability, through fault-tolerance, and higher performance through parallel processing.

RAID1: RAID level one, which corresponds to mirrored disks.

RAID3: RAID3 associates a parity disk to N data disks. Because all disks are written together, the parity can be computed on-the-fly and all disks can be read together to provide for a higher level of data integrity than provided by ECC alone.

RAID4: RAID4 is similar to RAID3 but has a larger striping unit. Data accesses in RAID4, as well as RAID5, are usually satisfied by a single disk access.

RAID5: RAID5 is RAID4 with rotated parity.

RAID6: A disk array utilizing two check disks. Reed-Solomon codes or P+Q coding is used to protect against two-disk failures.

Read-modify-write (RMW): The data is read, modified with the new data block or by computing the new parity blocks, and written after one disk rotation. Read-after-write verifies that the data has been written correctly.

SAN (storage area network): A network (switch) to which servers and storage peripherals are attached.

SCSI (Small Computer System Interface): Hardware and software standards for connecting peripherals to a computer system. Fast SCSI has an 8-bit bus and 10-MB/second transfer rate, while ULTRA320 has a 16-bit bus and 320-MB/second transfer rate.

Sector: The unit of track formatting and data transmission to/from disk drives. Sectors are usually 512 bytes long and are protected by an ECC.

Striping: A method of data allocation that partitions a dataset into equal-sized striping units, which are allocated in a round-robin manner on the disks of a disk array. This corresponds to RAID level zero.

Zero latency read: Ability to read data out of order so that latency is reduced, especially for accesses to larger blocks.

Zoned disk: The cylinders on the disk are partitioned into groups, called zones, which have a fixed number of sectors.

References

[Acharya et al. 1998] A. Acharya, M. Uysal, and J.H. Saltz. "Active disks: programming model, algorithms, and evaluation," *Proc. ASPLOS VIII*, 1998, pp. 81–91.

[Akyurek and Salem 1995] S. Akyurek and K. Salem. "Adaptive block rearrangement," *ACM Trans. Computer Systems*, 13(2):89–121, May 1995.

[Alvarez et al. 1997] G.A. Alvarez, W.A. Burkhard, and F. Cristian. "Tolerating multiple failures in RAID architectures with optimal storage and uniform declustering," *Proc. 24th ISCA*, 1997, pp. 62–72.

[Alvarez et al. 1998] G.A. Alvarez, W.A. Burkhard, L.J. Stockmeyer, and F. Cristian. "Declustered disk array architectures with optimal and near optimal parallelism," *Proc. 25th ISCA*, 1998, pp. 109–120.

[Alvarez et al. 2001] G.A. Alvarez et al. "Minerva: an automated resource provisioning tool for large-scale storage systems," *ACM Trans. Computer Systems*, 19(4):483–518 (2001).

[D. Anderson et al. 2003] D. Anderson, J. Dykes, and E. Riedel. "More than an interface — SCSI vs. ATA," *Proc. 2nd USENIX Conf. File and Storage Technologies*, USENIX, 2003, pp. 245–257.

[E. Anderson et al. 2002] E. Anderson et al. "Hyppodrome: Running circles around storage administration," *Proc. 1st Conf. on File and Storage Technologies — FAST '02*, USENIX, 2002, pp. 175–188.

[E. Anderson et al. 2003] E. Anderson et al. "Ergastulum: quickly finding near-optimal storage system designs," http:// www.hpl.hp.com/research/ssp/papers/ergastulum-paper.pdf

[Balafoutis et al. 2003] E. Balafoutis et al. "Clustered scheduling algorithms for mixed media disk workloads in a multimedia server," *Cluster Computing*, 6(1):75–86 (2003).

[Bennett and Franaszek 1977] B.T. Bennett and P.A. Franaszek. "Permutation clustering: an approach to online storage reorganization," *IBM J. Research and Development*, 21(6):528–533 (1977).

[Bitton and Gray 1988] D. Bitton and J. Gray. "Disk shadowing," *Proc. 14th Int. VLDB Conf.*, 1988, pp. 331–338.

[Blaum et al. 1995] M. Blaum, J. Brady, J. Bruck, and J. Menon. "EVENODD: an optimal scheme for tolerating disk failure in RAID architectures," *IEEE Trans. Computers*, 44(2):192–202 (Feb. 1995).

[Boxma and Cohen 1991] O.J. Boxma and J.W. Cohen. "The M/G/1 queue with permanent customers," *IEEE J. Selected Topics in Communications*, 9(2):179–184 (1991).
[Cao et al. 1994] P. Cao, S.B. Lim, S. Venkataraman, and J. Wilkes. "The TickerTAIP parallel RAID architecture," *ACM Trans. Computer Systems*, 12(3):236–269 (Aug. 1994).
[Chandy and Reddy 1993] J. Chandy and A.L. Narasimha Reddy. "Failure evaluation of disk array organizations," *Proc. 13th Int. Conf. on Distributed Computing Systems — ICDCS*, 1993, pp. 319–326.
[Chen et al. 1993] M.S. Chen, D.D. Kandlur, and P.S. Yu. "Optimization of the grouped sweeping scheduling for heterogeneous disks," *Proc. 1st ACM Int. Conf. on Multimedia*, 1993, pp. 235–242.
[Chen and Patterson 1990] P.M. Chen and D.A. Patterson. "Maximizing performance on a striped disk array," *Proc. 17th ISCA*, 1990, pp. 322–331.
[Chen et al. 1994] P.M. Chen, E.K. Lee, G.A. Gibson, R.H. Katz, and D.A. Patterson. "RAID: High-performance, reliable secondary storage," *ACM Computing Surveys*, 26(2):145–185 (1994).
[Chen and Patterson 1994] P.M. Chen and D.A. Patterson. "A new approach to I/O performance evaluation: self-scaling I/O benchmarks, predicted I/O performance," *ACM Trans. Computer Systems*, 12(4):308–339 (Nov. 1994).
[Chen and Towsley 1993] S.-Z. Chen and D.F. Towsley. "The design and evaluation of RAID5 and parity striping disk array architectures," *J. Parallel and Distributed Computing*, 10(1/2):41–57 (1993).
[Chen and Towsley 1996] S.-Z. Chen and D.F. Towsley. "A performance evaluation of RAID architectures," *IEEE Trans. Computers*, 45(10):1116–1130 (1996).
[Coffman et al. 1972] E.G. Coffman Jr., E.G. Klimko, and B. Ryan. "Analyzing of scanning policies for reducing disk seek times," *SIAM J. Computing*, 1(3):269–279 (1972).
[Coffman and Hofri 1982] E.G. Coffman, Jr. and M. Hofri. "On the expected performance of scanning disks," *SIAM J. Computing*, 11(1):60–70 (1982).
[Coffman and Hofri 1990] E.G. Coffman, Jr. and M. Hofri. "Queueing models of secondary storage devices," in *Stochastic Analysis of Computer and Communication Systems*, H. Takagi (Ed.), North-Holland, 1990, pp. 549–588.
[Cortes and Labarta 2001] T. Cortes and J. Labarta. "Extending heterogeneity to RAID level 5," *Proc. USENIX Annual Technical Conf.*, 2001, pp. 119–132.
[Courtright et al. 1996] W.V. Courtright II et al. "RAIDframe: A Rapid Prototyping Tool for Raid Systems," http://www.pdl.cmu.edu/RAIDframe/raidframebook.pdf.
[Courtright 1997] W.V. Courtright II. "A Transactional Approach to Redundant Disk Array Implementation," Technical Report CMU-CS-97-141, 1997.
[Denning 1967] P.J. Denning. "Effects of scheduling in file memory operations," *Proc. AFIPS Spring Joint Computer Conf.*, 1967, pp. 9–21.
[Diskspecs] http://www.pdl.cmu.edu/DiskSim/diskspecs.html.
[DiskSim] http://www.pdl.cmu.edu/DiskSim/disksim2.0.html.
[Dowdy and Foster 1982] L.W. Dowdy and D.V. Foster. "Comparative models of the file assignment problem," *ACM Computing Surveys*, 14(2):287–313 (1982).
[Geist and Daniel 1987] R.M. Geist and S. Daniel. "A continuum of disk scheduling algorithm," *ACM Trans. Computer Systems*, 5(1):77–92 (1987).
[Gibson 1992] G.A. Gibson. *Redundant Disk Arrays: Reliable, Parallel Secondary Storage*, The MIT Press, 1992.
[Gibson et al. 1995] G.A. Gibson et al. "The Scotch parallel storage system," *Proc. IEEE CompCon Conf.*, 1995, pp. 403–410.
[Gibson et al. 1998] G.A. Gibson et al. "A cost-effective, high bandwidth storage architecture," *Proc. ASPLOS VIII*, 1998, 92–103.
[Gibson and Van Meter 2002] G.A. Gibson and R. Van Meter. "Network attached storage architecture," *Commun. ACM*, 43(11):37–45 (Nov. 2000).
[Gray and Reuter 1993] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques*, Morgan-Kaufmann Publishers, 1993.

[Gray et al. 1990] J. Gray, B. Horst, and M. Walker. "Parity striping of disk arrays: low-cost reliable storage with acceptable throughput," *Proc. 16th Int. VLDB Conf.*, 1990, 148–161.

[Gray and Shenoy 2000] J. Gray and P. J. Shenoy. "Rules of thumb in data engineering," *Proc. 16th ICDE*, 2000, pp. 3–16.

[Gray 2002] J. Gray. "Storage bricks have arrived" (Keynote Speech), *1st Conf. on File and Storage Technologies-FAST '02*, USENIX, 2002.

[Griffin et al. 2000] J. L. Griffin, S. W. Shlosser, G. R. Ganger, and D. F. Nagle. "Modeling and performance of MEMS-based storage devices," *Proc. ACM SIGMETRICS 2000*, pp. 56–65.

[Gurumurthi et al. 2003] S. Gurumurthi, A. Sivasubramaniam, M. Kandemir, and H. Franke. "DRPM: Dynamic speed control for power management in server-class disks," *Proc. 30th ISCA*, 2003, pp. 169–181.

[Hall 1986] M. Hall. *Combinatorial Theory*, Wiley, 1986.

[Han and Thomasian 2003] C. Han and A. Thomasian. "Performance of two-disk failure tolerant disk arrays," *Proc. Symp. Performance Evaluation of Computer and Telecomm. Systems — SPECTS '03*, 2003.

[Hartman and Ousterhout 1995] J.H. Hartman and J.K. Ousterhout. "The Zebra striped network file system," *ACM Trans. Computer Systems*, 13(3):274–310 (1995).

[Hellerstein et al. 1994] L. Hellerstein et al. "Coding techniques for handling failures in large disk arrays," *Algorithmica*, 12(2/3):182–208 (Aug./Sept. 1994).

[Hennessey and Patterson 2003] J. Hennessey and D. Patterson. *Computer Organization: A Quantitative Approach, 3rd ed.* Morgan-Kauffman Publishers, 2003.

[Holland et al. 1994] M.C. Holland, G.A. Gibson, and D.P. Siewiorek. "Architectures and algorithms for on-line failure recovery in redundant disk arrays," *Distributed and Parallel Databases*, 11(3):295–335 (1994).

[Hsiao and DeWitt 1993] H.I. Hsiao and D.J. DeWitt. "A performance study of three high availability data replication strategies," *J. Distributed and Parallel Databases*, 1(1):53–80 (Jan. 1993).

[Hsu et al. 2001] W.W. Hsu, A.J. Smith, and H.C. Young. "I/O reference behavior of production database workloads and the TPC benchmarks — an analysis at the logical level," *ACM Trans. Database Systems*, 26(1): 96–143 (2001).

[Hsu and Smith 2003] W.W. Hsu and A.J. Smith. "Characteristics of I/O traffic in personal computer and server workloads," *IBM Systems J.*, 42(2):347–372 (2003).

[Iyer and Druschel 2001] S. Iyer and P. Druschel. "Anticipatory scheduling: a disk scheduling framework to overcome deceptive idleness in synchronous I/O," *Proc. 17th SOSP*, 2001, pp. 117–130.

[Jacobson and Wilkes 1991] D. Jacobson and J. Wilkes. "Disk scheduling algorithms based on rotational position," *HP Technical Report* HPL-CSP-91-7rev, 1991.

[Jain et al. 1996] R. Jain, J. Werth, and J.C. Browne (Editors). *Input/Output in Parallel and Distributed Systems*, Kluwer Academic Publishers, 1996.

[Jin et al. 2002] H. Jin, T. Cortes, and R. Buyya. *High Performance Mass Storage and Parallel I/O: Technologies and Applications*, Wiley Interscience, 2002.

[King 1990] R.P. King. "Disk arm movement in anticipation of future requests," *ACM Trans. Computer Systems*, 8(3):214–229 (1990).

[Kleinrock 1975] L. Kleinrock. *Queueing Systems, Vol. I: Theory*, Wiley Interscience, 1975.

[Kleinrock 1976] L. Kleinrock. *Queueing Systems, Vol. II: Computer Applications*, Wiley Interscience, 1976.

[Kozyrakis and Patterson 2003] C. Kozyrakis and D. Patterson. "Overcoming the limitations of current vector processors," *Proc. 30th ISCA*, 2003, 399–409.

[Lawlor 1981] F.D. Lawlor. "Efficient mass storage parity recovery mechanism," *IBM Technical Disclosure Bulletin*, 24(2):986–987 (July 1981).

[Lazowska et al. 1984] E.D. Lazowska, J. Zahorjan, G.S. Graham, and K.C. Sevcik. *Quantitative System Performance*, Prentice Hall, 1984. Also http://www.cs.washington.edu/homes/lazowska/qsp.

[Lee and Thekkath 1996] E.K. Lee and C. Thekkath. "Petal: distributed virtual disks," *Proc. ASPLOS XII*, 1996, pp. 84–92.

[Lee and Katz 1993] E.K. Lee and R.H. Katz. "The performance of parity placements in disk arrays," *IEEE Trans. Computers*, 42(6):651–664 (June 1993).

[Lumb et al. 2000] C.R. Lumb, J. Schindler, G.R. Ganger, and D.F. Nagle. "Towards higher disk head utilization: extracting free bandwidth from busy disk drives," *Proc. 4th OSDI Symp.* USENIX, 2000, pp. 87–102.

[Lynch 1972] W.C. Lynch. "Do disk arms move?," *Performance Evaluation Review*, 1(4):3–16 (Dec. 1972).

[MacWilliams and Sloane 1977] F.J. MacWilliams and N.J.A. Sloane. *The Theory of Error Correcting Codes*, North-Holland, 1977.

[Matick 1977] R.E. Matick. *Computer Storage Systems and Technology*, Wiley, 1977.

[McKusick et al. 1984] M.K. McKusick, W.N. Joy, S.J. Leffler, and R.S. Fabry. "A fast file system for UNIX," *ACM Trans. Computer Systems*, 2(3):181–197 (1984).

[Menon and Cortney 1993] J. Menon and J. Cortney. "The architecture of a fault-tolerant cached RAID controller," *Proc. 20th ISCA*, 1993, pp. 76–86.

[Menon 1994] J. Menon. "Performance of RAID5 disk arrays with read and write caching," *Distributed and Parallel Databases*, 11(3):261–293 (1994).

[Menon 1995] J. Menon. "A performance comparison of RAID5 and log-structured arrays," *Proc. 4th IEEE HPDC*, 1995, pp. 167–178.

[Merchant and Yu 1996] A. Merchant and P.S. Yu. "Analytic modeling of clustered RAID with mapping based on nearly random permutation," *IEEE Trans. Computers*, 45(3):367–373 (1996).

[Meritt et al. 2003] A.S. Meritt et al. "z/OS support for IBM TotalStorage enterprise storage server," *IBM Systems J.*, 42(2):280–301 (2003).

[Mesnier et al. 2003] M. Mesnier, G.R. Ganger, and E. Riedel. "Object-based storage," *IEEE Communications Magazine*, 41(8):84–90 (2003).

[Meth and Satran 2003] K.Z. Meth and J. Satran. "Features of the iSCSI protocol," *IEEE Communications Magazine*, 41(8):72–75 (2003).

[Muntz and Lui 1990] R. Muntz and J.C.S. Lui. "Performance analysis of disk arrays under failure," *Proc. 16th Int. VLDB Conf.*, 1990, pp. 162–173.

[Nelson and Tantawi 1988] R. Nelson and A. Tantawi. "Approximate analysis of fork-join synchronization in parallel queues," *IEEE Trans. Computers*, 37(6):739–743 (1988).

[Newberg and Wolfe 1994] L. Newberg and D. Wolfe. "String layout for a redundant array of inexpensive disks," *Algorithmica*, 12(2/3):209–224 (Aug./Sept. 1994).

[Ng 1987] S.W. Ng. "Reliability, availability, and performance analysis of duplex disk systems," *Reliability and Quality Control*, M.H. Hamza (Ed.), Acta Press, 1987, pp. 5–9.

[Ng 1991] S.W. Ng. "Improving disk performance via latency reduction," *IEEE Trans. Computers*, 40(1):22–30 (Jan. 1991).

[Ng and Mattson 1994] S.W. Ng and R.L. Mattson. "Uniform parity distribution in disk arrays with multiple failures," *IEEE Trans. Computers*, 43(4):501–506 (1994).

[Ng 1994] S.W. Ng. "Crosshatch disk array for improved reliability and performance," *Proc. 21st ISCA*, 1994, pp. 255–264.

[Ng 1998] S.W. Ng. "Advances in disk technology: performance issues," *IEEE Computer*, 40(1):75–81 (1998).

[Patterson et al. 1998] D.A. Patterson, G.A. Gibson, and R.H. Katz. "A case study for redundant arrays of inexpensive disks," *Proc. ACM SIGMOD Int. Conf.*, 1998, pp. 109–116.

[Patterson et al. 1995] R.H. Patterson et al. "Informed prefetching and caching," *Proc. 15th SOSP*, 1995, pp. 79–95.

[Pfister 2002] G.F. Pfister. "An introduction to the InfiniBand™ architecture," in *High Performance Mass Storage and Parallel I/O*, H. Jin et al. (Eds.), Wiley, 2002, pp. 617–632.

[Polyzois et al. 1993] C. Polyzois, A. Bhide, and D.M. Dias. "Disk mirroring with alternating deferred updates," *Proc. 19th Int. VLDB Conf.*, 1993, 604–617.

[Ramakrishnan et al. 1992] K.K. Ramakrishnan, P. Biswas, and R. Karedla. "Analysis of file I/O traces in commercial computing environments," *Proc. Joint ACM SIGMETRICS/Performance '92 Conf.*, 1992, pp. 78–90.

[Ramakrishnan and Gehrke 2003] R. Ramakrishnan and J. Gehrke. *Database Management Systems, 3rd ed.*, McGraw-Hill, 2003.

[Riedel et al. 2000] E. Riedel, C. Faloutsos, G.R, Ganger, and D.F. Nagle. "Data mining in an OLTP system (nearly) for free," *Proc. ACM SIGMOD Int. Conf.*, 2000, pp. 13–21.

[Riedel et al. 2001] E. Riedel, C. Faloutsos, G.R. Ganger, and D.F. Nagle. "Active disks for large scale data processing," *IEEE Computer*, 34(6):68–74 (2001).

[Rosenblum and Ousterhout 1992] M. Rosenblum and J.K. Ousterhout. "The design and implementation of a log-structured file system," *ACM Trans. Computer Systems*, 10(1):26–52 (Feb. 1992).

[Ruemmler and Wilkes 1992] C. Ruemmler and J. Wilkes. "UNIX disk access patterns," *HP Labs Technical Report*, HPL-92-152, 1992.

[Ruemmler and Wilkes 1993] C. Ruemmler and J. Wilkes. "A trace-driven analysis of disk working set sizes," *HP Labs Technical Report*, HPL-93-23, 1993.

[Ruemmler and Wilkes 1994] C. Ruemmler and J. Wilkes. "An introduction to disk drive modeling," *IEEE Computer*, 27(3):17–28 (March 1994).

[Salzwedel 2003] K.A. Salzwedel. "Algorithmic approaches for storage networks," in *Algorithms for Memory Hierarchies, Advanced Lectures LNCS 2625*, Springer, 2003, pp. 251–272.

[Santos et al. 2000] J.R. Santos, R.R. Muntz, and B. Ribeiro-Neto. "Comparing random data allocation and data striping in multimedia servers," *Proc. ACM SIGMETRICS Conf.*, 2000, pp. 44–55.

[Sarkar et al. 2003] P. Sarkar, S. Uttamchandani, and K. Voruganti. "Storage over IP: when does hardware support help?," *Proc. 2nd Conf. on File and Storage Technologies — FAST '03*, USENIX, 2003.

[Schindler and Ganger 1999] J. Schindler and G. R. Ganger. "Automated disk drive characterization," *CMU SCS Technical Report*, CMU-CS-99-176, 1999.

[Schindler et al. 2002] J. Schindler, J.L. Griffin, C.R. Lumb, and G.R. Ganger. "Track-aligned extents: matching access patterns to disk drive characteristics," *Proc. Conf. on File and Storage Technologies — FAST '02*, USENIX, 2002.

[Schwartz 1999] T.J.E. Schwartz, J. Steinberg, and W.A. Burkhard. "Permutation development data layout (PDDL) disk array declustering," *Proc. 5th IEEE Symp. on High Performance Computer Architecture — HPCA*, 1999, pp. 214–217.

[Seltzer et al. 1990] M.I. Seltzer, P.M. Chen, and J.K. Ousterhout. "Disk scheduling revisited," *Proc. 1990 USENIX Summer Technical Conf.*, pp. 307–326.

[Shriver et al. 2000] E. Shriver, B.K. Hillyer, and A. Silberschatz. "Performance analysis of storage systems," in *Performance Evaluation: Origins and Directions — LNCS 1769*, G. Haring, C. Lindemann, and M. Reiser (Editors), Springer-Verlag, 2000.

[Sierra 1990] H.M. Sierra. *An Introduction to Direct Access Storage Devices*, Academic Press, 1990.

[Sitaram and Dan 2000] D. Sitaram and A. Dan. *Multimedia Servers: Applications, Environments, and Design*, Morgan-Kaufmann Publishers, 2000.

[Smith 1985] A.J. Smith. "Disk cache: miss ratio analysis and design considerations," *ACM Trans. Computer Systems*, 3(3):161–203 (Aug. 1985).

[Solworth and Orji 1991] J.A. Solworth and C.U. Orji. "Distorted mirrors," *Proc. 1st Int. Conf. on Parallel and Distributed Information Systems — PDIS*, 1991, pp. 10–17.

[Stodolsky et al. 1994] D. Stodolsky, M. Holland, W.C. Courtright II, and G.A. Gibson. "Parity logging disk arrays," *ACM Trans. Computer Systems*, 12(3):206–325 (1994).

[Storage Performance Council 2003] http://www.storageperformance.org.

[Takagi 1991] H. Takagi. *Queueing Analysis. Vol. 1: Vacation and Priority Systems*, North-Holland, 1991.

[Thomasian and Menon 1994] A. Thomasian and J. Menon. "Performance analysis of RAID5 disk arrays with a vacationing server model," *Proc. 10th ICDE Conf.*, 1994, pp. 111–119.

[Thomasian and Menon 1997] A. Thomasian and J. Menon. "RAID5 performance with distributed sparing," *IEEE Trans. Parallel and Distributed Systems*, 8(6):640–657 (June 1997).
[Thomasian 1997] A. Thomasian. "Approximate analysis for fork-join synchronization in RAID5," *Computer Systems: Science and Engineering*, 12(5):329–338 (1997).
[Thomasian 1998] A. Thomasian. "RAID5 disk arrays and their performance evaluation," in *Recovery Mechanisms in Database Systems*, V. Kumar and M. Hsu (Eds.), 1998 pp. 807–846.
[Thomasian and Liu 2002a] A. Thomasian and C. Liu. "Some new disk scheduling policies and their performance," *Proc. ACM SIGMETRICS Conf.*, 2002, pp. 266–267.
[Thomasian and Liu 2002b] A. Thomasian and C. Liu. "Disk scheduling policies with lookahead," *Performance Evaluation Review*, 30(2):31–40 (Sept. 2002).
[Thomasian et al. 2003] A. Thomasian et al. "Mirrored disk scheduling," *Proc. Symp. Performance Evaluation of Computer and Telecomm. Systems — SPECTS '03*, 2003.
[Thomasian and Liu 2003] A. Thomasian and C. Liu. "Performance of mirrored disks with a shared NVS cache," submitted for publication (available from authors).
[Toigo 2000] J.W. Toigo. *The Holy Grail of Data Storage Management*, Prentice-Hall, 2000.
[Treiber and Menon 1995] K. Treiber and J. Menon. "Simulation study of cached RAID5 designs," *Proc. 1st IEEE Symp. on High Performance Computer Architecture — HPCA*, 1995, pp. 186–197.
[Trivedi 2002] K.S. Trivedi. *Probability and Statistics with Reliability, Queueing and Computer Science Applications, 2nd edition*, Wiley Interscience, 2002.
[Uysal et al. 2001] M. Uysal, G.A. Alvarez, and A. Merchant. "A modular, analytical throughput model for modern disk drives," *Proc. MASCOTS-2001*.
[Uysal et al. 2003] M. Uysal, A. Merchant, and G.A. Alvarez. "Using MEMS-based storage in disk arrays," *Proc. 2nd Conf. on File and Storage Technologies — FAST '03*, USENIX, 2003.
[Varma and Jacobson 1998] A. Varma and Q. Jacobson. "Destage algorithms for disk arrays with non-volatile caches," *IEEE Trans. Computers*, 47(2):228–235 (1998).
[Varma and Makowski 1994] S. Varma and A. Makowski. "Interpolation approximations for symmetric fork-join queues," *Performance Evaluation*, 20(1–3):361–368 (1994).
[Wilkes et al. 1996] J. Wilkes, R.A. Golding, C. Staelin, and T. Sullivan. "The HP AutoRAID hierarchical storage system," *ACM Trans. Computer Systems*, 14(1):108–136 (1996).
[Wilkes 2001] J. Wilkes. "Traveling to Rome: QoS specifications for automated storage management," *Proc. Int. Workshop on Quality of Service — IWQOS*, Springer-Verlag, 2001.
[Wilkes 2003] J. Wilkes. "Data services — from data to containers," *2nd Conf. on File and Storage Technologies — FAST '03*, USENIX, (Keynote Speech). http://www.usenix.com/publications/library/proceedings/ fast03/tech.html
[Wolf 1989] J.L. Wolf. "The placement optimization program: a practical solution to the disk file assignment problem," *Proc. ACM SIGMETRICS Conf.*, 1989, pp. 1–10.
[Wong 1980] C.K. Wong. "Minimizing head movement in one dimensional and two dimensional mass storage systems," *ACM Computing Surveys*, 12(2):167–178 (1980).
[Wong 1983] C.K. Wong. *Algorithmic Studies in Mass Storage Systems*, Computer Science Press, 1983.
[Worthington et al. 1994] B.L. Worthington, G.R. Ganger, and Y.L. Patt. "Scheduling for modern disk drivers and non-random workloads," *Proc. ACM SIGMETRICS Conf. 1994*, pp. 241–251.
[Worthington et al. 1995] B.L. Worthington, G.R. Ganger, Y.N. Patt, and J. Wilkes. "On-line extraction of SCSI disk drive parameters," *Proc. 1995 ACM SIGMETRICS Conf.*, pp. 146–156 (also *HP Labs Technical Report* HPL-97-02, 1997).
[Wylie et al. 2000] J.J. Wylie et al. "Survivable information storage systems," *IEEE Computer*, 33(8):61–68 (Aug. 2000).
[Zhang et al. 2002] C. Zhang, X. Yu, A. Krishnamurthy, and R.Y. Wang. "Configuring and scheduling an eager-writing disk array for a transaction processing workload," *Proc. 1st Conf. on File and Storage Technologies — FAST '02*, USENIX, 2002, pp. 289–304.

Further Information

Chapters with overlapping content, whose topics should be of interest to the readers of this chapter, include the following:

[**Access Methods by B. Salzberg**]. Especially Section 1, Introduction: brief description of magnetic disks.

[**Process and Device Scheduling by R.D. Cupper**]. Especially Section 6, Device Scheduling. The discussion of Scheduling Shared Devices (disks) is not as complete as our discussion and SPTF is not mentioned at all.

[**Secondary Storage and Filesystems by M. K. McKusick**]. Especially Section 1 Introduction: Computer Storage Hierarchy. Section 2 Secondary Storage Devices: Magnetic disks, RAID. Section 3 Filesystems: Directory structure, file layout on disk, file transfers, disk space management, logging, including LFS, file allocation, and disk defragmentation.

In addition to Gibson [1992], two useful books are: J.W. Toigo, *The Holy Grail of Data Storage Management*, Addison-Wesley, 2000 (an overview of the field for a nontechnical reader) and H. Jin, T. Cortes, and R. Buyya, *High Performance Mass Storage and Parallel I/O: Technologies and Applications*, Wiley, 2002 (a collection of papers, with emphasis on parallel I/O).

Matick [1977] discusses early computer storage systems from hardware and software viewpoint. Sierra [1990] is a text dedicated to disk technology, but is somewhat dated.

The *IBM Systems J.*, Vol. 42, No. 2, 2003, is on *Storage Systems*. Similar information is available as "White Papers" from the Web pages of EMC, Hitachi, HP, etc.

Disk drive characteristics are available from the Web sites of disk drive manufacturers: Hitachi, Maxtor, Seagate, Western Digital, etc.

The *File and Storage Technologies Conf.* organized by USENIX and the *Mass Storage Systems Conf.* co-sponsored by NASA and IEEE are two specialized conferences.

Some web sites of interest are as follows:
HP Labs Storage Systems Program: http://www.hpl.hp.com/research/ssp
CMU's Parallel Data Laboratory: http://www.pdl.cmu.edu
UC Berkeley: http://roc.cs.berkeley.edu
UC Santa Cruz: http://csl.cse.ucsc.edu
U. Wisconsin: http://www.cs.wisc.edu/wind

22

High-Speed Computer Arithmetic

22.1 Introduction

22.2 Fixed Point Number Systems
Two's Complement • Sign Magnitude • One's Complement

22.3 Fixed Point Arithmetic Algorithms
Fixed Point Addition • Fixed Point Subtraction • Fixed Point Multiplication • Fixed Point Division

22.4 Floating Point Arithmetic
Floating Point Number Systems

22.5 Conclusion

Earl E. Swartzlander Jr.
University of Texas at Austin

22.1 Introduction

The speeds of memory and arithmetic units are the primary determinants of the speed of a computer. Whereas the speed of both units depends directly on the implementation technology, arithmetic unit speed also depends strongly on the logic design. Even for an integer adder, speed can easily vary by an order of magnitude, whereas the complexity varies by less than 50%.

This chapter begins with a discussion of binary fixed point number systems in Section 22.2. Section 22.3 provides examples of fixed point implementations of the four basic arithmetic operations (i.e., add, subtract, multiply, and divide). Finally, Section 22.4 describes algorithms that implement floating point arithmetic.

Regarding notation, capital letters represent digital numbers (i.e., words), whereas subscripted lowercase letters represent bits of the corresponding word. The subscripts range from $n-1$ to 0, to indicate the bit position within the word (x_{n-1} is the most significant bit of X, x_0 is the least significant bit of X, etc.). The logic designs presented in this chapter are based on positive logic with AND, OR, and INVERT operations. Depending on the technology used for implementation, different operations (such as NAND and NOR) can be used, but the basic concepts are not likely to change significantly.

22.2 Fixed Point Number Systems

Most arithmetic is performed with fixed point numbers that have constant scaling (i.e., the position of the binary point is fixed). The numbers can be interpreted as fractions, integers, or mixed numbers, depending on the application. Pairs of fixed point numbers are used to create floating point numbers, as discussed in Section 22.4.

At the present time, fixed point binary numbers are generally represented using the two's complement number system. This choice has prevailed over the sign magnitude and one's complement number systems, because the frequently performed operations of addition and subtraction are more easily performed on two's complement numbers. Sign magnitude numbers are more efficient for multiplication but the lower frequency of multiplication and the development of Booth's efficient two's complement multiplication algorithm have resulted in the nearly universal selection of the two's complement number system for most applications. The algorithms presented in this chapter assume the use of two's complement numbers.

Fixed point number systems represent numbers, for example, A, by n bits: a sign bit and $n-1$ data bits. By convention, the most significant bit a_{n-1} is the sign bit, which is a 1 for negative numbers and a 0 for positive numbers. The $n-1$ data bits are $a_{n-2}, a_{n-3}, \ldots, a_1, a_0$. In the following material, fixed point fractions will be described for each of the three systems.

22.2.1 Two's Complement

In the two's complement fractional number system, the value of a number is the sum of $n-1$ positive binary fractional bits and a sign bit, which has a weight of -1:

$$A = -a_{n-1} + \sum_{i=0}^{n-2} a_i 2^{i-n+1} \tag{22.1}$$

Two's complement numbers are negated by complementing all bits and adding a 1 to the least significant bit (lsb) position. For example, to form $-3/8$,

$$\begin{aligned} +3/8 &= 0011 \\ \text{invert all bits} &= 1100 \\ \text{add 1 lsb} \quad & 0001 \\ & 1101 = -3/8 \end{aligned}$$

Check:

$$\begin{aligned} \text{invert all bits} &= 0010 \\ \text{add 1 lsb} \quad & 0001 \\ & 0011 = +3/8 \end{aligned}$$

22.2.2 Sign Magnitude

Sign magnitude numbers consist of a sign bit and $n-1$ bits that express the magnitude of the number,

$$A = (1 - 2a_{n-1}) \sum_{i=0}^{n-2} a_i 2^{i-n+1} \tag{22.2}$$

Sign magnitude numbers are negated by complementing the sign bit. For example, to form $-3/8$,

$$\begin{aligned} +3/8 &= 0011 \\ \text{invert sign bit} &= 1011 = -3/8 \end{aligned}$$

Check:

$$\text{invert sign bit} = 0011 = +3/8$$

22.2.3 One's Complement

One's complement numbers are negated by complementing all of the bits of the original number,

$$A = \sum_{i=0}^{n-2} (a_i - a_{n-1}) 2^{i-n+1} \tag{22.3}$$

In this equation, the subtraction $(a_i - a_{n-1})$ is an arithmetic operation (not a logical operation) that produces values of 1 or 0 (if $a_{n-1} = 0$) or values of 0 or -1 (if $a_{n-1} = 1$).

The negative of a one's complement number is formed by inverting all bits. For example, to form $-3/8$,

$$+3/8 = 0011$$
$$\text{invert all bits} = 1100 = -3/8$$

Check:

$$\text{invert all bits} = 0011 = +3/8$$

Table 22.1 compares 4-bit fractional fixed point numbers in the three number systems. Note that both the sign magnitude and one's complement number systems have two zeros (i.e., positive zero and negative zero) and that only two's complement is capable of representing -1. For positive numbers, all three number systems use identical representations.

A significant difference between the three number systems is their behavior under truncation. Figure 22.1 shows the effect of truncating high-precision fixed point fractions X, to form three bit fractions $T(X)$. Truncation of two's complement numbers never increases the value of the number (i.e., the truncated numbers have values that are unchanged or shift toward negative infinity), as can be seen from Equation 22.1 since any truncated bits have positive weight. This bias can cause an accumulation of errors for computations that involve summing many truncated numbers (which may occur in scientific, matrix, and signal processing applications). In both the sign magnitude and one's complement number systems, truncated numbers are unchanged or shifted toward zero, so that if approximately half of the numbers are positive and half are negative, the errors will tend to cancel.

TABLE 22.1 Example of 4-bit Fractional Fixed Point Numbers

Number	Two's Complement	Sign Magnitude	One's Complement
$+7/8$	0111	0111	0111
$+3/4$	0110	0110	0110
$+5/8$	0101	0101	0101
$+1/2$	0100	0100	0100
$+3/8$	0011	0011	0011
$+1/4$	0010	0010	0010
$+1/8$	0001	0001	0001
$+0$	0000	0000	0000
-0	N/A	1000	1111
$-1/8$	1111	1001	1110
$-1/4$	1110	1010	1101
$-3/8$	1101	1011	1100
$-1/2$	1100	1100	1011
$-5/8$	1011	1101	1010
$-3/4$	1010	1110	1001
$-7/8$	1001	1111	1000
-1	1000	N/A	N/A

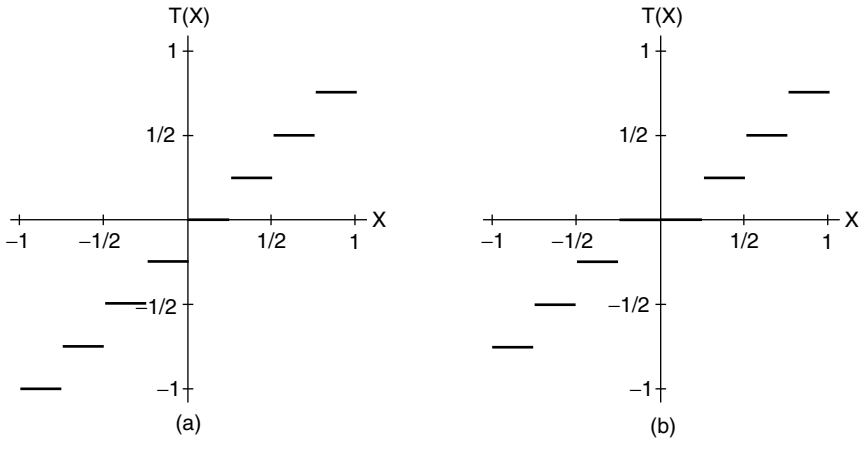

FIGURE 22.1 Behavior of fixed point fractions under truncation: (a) two's complement and (b) sign magnitude and one's complement.

TABLE 22.2 Full Adder Truth Table

Inputs			Outputs	
a_k	b_k	c_k	c_{k+1}	s_k
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

22.3 Fixed Point Arithmetic Algorithms

This section presents an assortment of typical fixed point algorithms for addition, subtraction, multiplication, and division.

22.3.1 Fixed Point Addition

Addition is performed by summing the corresponding bits of two n-bit numbers, including the sign bit. Subtraction is performed by summing the corresponding bits of the minuend and the two's complement of the subtrahend. Overflow is detected in a two's complement adder by comparing the carry signals into and out of the most significant adder stage (i.e., the stage which computes the sign bit). If the carries differ, an overflow has occurred and the result is invalid.

22.3.1.1 Full Adder

The full adder is the fundamental building block of most arithmetic circuits. Its operation is defined by the truth table shown in Table 22.2. The sum and the carry outputs are described by the following equations:

$$s_k = a_k \bar{b}_k \bar{c}_k + \bar{a}_k b_k \bar{c}_k + \bar{a}_k \bar{b}_k c_k + a_k b_k c_k = a_k \oplus b_k \oplus c_k \tag{22.4}$$

$$c_{k+1} = \bar{a}_k b_k c_k + a_k \bar{b}_k c_k + a_k b_k \bar{c}_k + a_k b_k c_k = a_k b_k + a_k c_k + b_k c_k \tag{22.5}$$

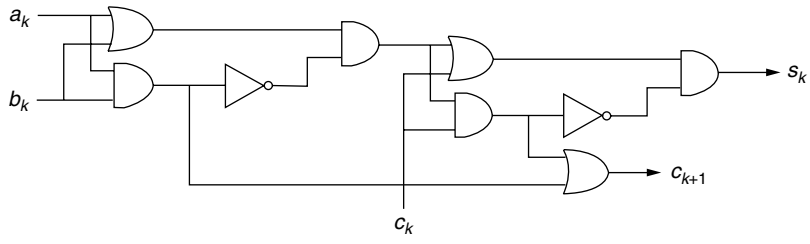

FIGURE 22.2 9-gate full adder.

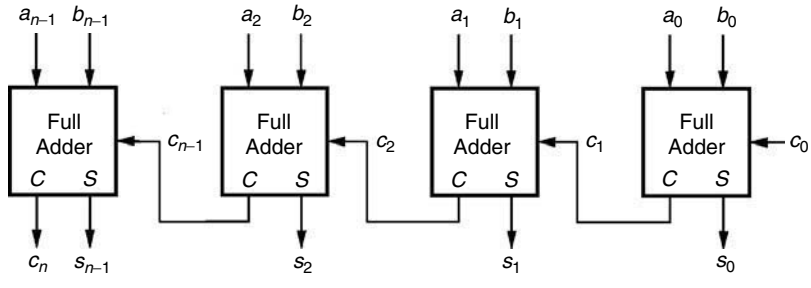

FIGURE 22.3 Ripple carry adder.

where a_k, b_k, and c_k are the inputs to the kth full adder stage, and s_k and c_{k+1} are the sum and carry outputs, respectively.

In evaluating the relative complexity of implementations, it is often convenient to assume a nine gate realization of the full adder, as shown in Figure 22.2. For this implementation, the delay from either a_k or b_k to s_k is six gate delays and the delay from c_k to c_{k+1} is two gate delays. Some technologies, such as CMOS, form inverting gates (e.g., NAND and NOR gates) more efficiently than the noninverting gates that are assumed in this chapter. Circuits with equivalent speed and complexity can be constructed with inverting gates.

22.3.1.2 Ripple Carry Adder

A ripple carry adder for n-bit numbers is implemented by concatenating n full adders as shown in Figure 22.3. At the kth-bit position, the kth bits of operands A and B and a carry signal from the preceding adder stage are used to form the kth bit of the sum, s_k, and the carry, c_{k+1}, to the next adder stage. This is called a ripple carry adder, since the carry signals *ripple* from the least significant bit position to the most significant bit position. If the ripple carry adder is implemented by concatenating n of the nine gate full adders, as shown in Figure 22.2 and Figure 22.3, an n-bit ripple carry adder requires $2n + 4$ gate delays to produce the most significant sum bit and $2n + 3$ gate delays to produce the carry output. A total of $9n$ logic gates are required to implement an n-bit ripple carry adder.

In comparing adders, the delay from data input to the slowest (usually the most significant sum) output and the complexity (i.e., the gate count) will be used. These will be denoted by DELAY and GATES (subscripted by RCA to indicate ripple carry adder), respectively. These simple metrics are suitable for first-order comparisons. More accurate comparisons require consideration of both the number and the types of gates (because gates with fewer inputs are generally faster and smaller than gates with more inputs).

$$\text{DELAY}_{\text{RCA}} = 2n + 4 \tag{22.6}$$

$$\text{GATES}_{\text{RCA}} = 9n \tag{22.7}$$

22.3.1.3 Carry Lookahead Adder

Another approach is the carry lookahead adder [Weinberger and Smith 1958, MacSorley 1961]. Here, specialized carry logic computes the carries in parallel. The carry lookahead adder uses MFAs, or modified full adders (modified in the sense that a carry output is not formed) for each bit position and lookahead modules. Each lookahead module forms individual carry outputs and also block generate and block propagate outputs that indicate that a carry is generated within the module or that an incoming carry would propagate across the module, respectively. Rewriting Equation 22.5 using $g_k = a_k b_k$ and $p_k = a_k + b_k$:

$$c_{k+1} = g_k + p_k c_k \tag{22.8}$$

This explains the concept of carry generation and carry propagation. At a given stage, a carry is generated if g_k is true (i.e., both a_k and b_k are 1), and a stage propagates a carry from its input to its output if p_k is true (i.e., either a_k or b_k is a 1). The nine gate full adder shown in Figure 22.2 has AND and OR gates that produce g_k and p_k with no additional complexity. Because the carry out is produced by the lookahead logic, the OR gate that produces c_{k+1} can be eliminated. The result is an eight gate modified full adder (MFA). Extending Equation 22.8 to a second stage,

$$\begin{aligned} c_{k+2} &= g_{k+1} + p_{k+1} c_{k+1} \\ &= g_{k+1} + p_{k+1}(g_k + p_k c_k) \\ &= g_{k+1} + p_{k+1} g_k + p_{k+1} p_k c_k \end{aligned} \tag{22.9}$$

This equation results from evaluating Equation 22.8 for the $k+1$st stage and substituting c_{k+1} from Equation 22.8. Carry c_{k+2} exits from stage $k+1$ if: (1) a carry is generated there; or (2) a carry is generated in stage k and propagates across stage $k+1$; or (3) a carry enters stage k and propagates across both stages k and $k+1$. Extending to a third stage,

$$\begin{aligned} c_{k+3} &= g_{k+2} + p_{k+2} c_{k+2} \\ &= g_{k+2} + p_{k+2}(g_{k+1} + p_{k+1} g_k + p_{k+1} p_k c_k) \\ &= g_{k+2} + p_{k+2} g_{k+1} + p_{k+2} p_{k+1} g_k + p_{k+2} p_{k+1} p_k c_k \end{aligned} \tag{22.10}$$

Although it would be possible to continue this process indefinitely, each additional stage increases the size (i.e., the number of inputs) of the logic gates. Four inputs (as required to implement Equation 22.10) is frequently the maximum number of inputs per gate for current technologies. To continue the process, generate and propagate signals are defined over four bit blocks (stages k to $k+3$), $g_{k+3:k}$ and $p_{k+3:k}$, respectively.

$$g_{k+3:k} = g_{k+3} + p_{k+3} g_{k+2} + p_{k+3} p_{k+2} g_{k+1} + p_{k+3} p_{k+2} p_{k+1} g_k \tag{22.11}$$

and

$$p_{k+3:k} = p_{k+3} p_{k+2} p_{k+1} p_k \tag{22.12}$$

Equation 22.8 can be expressed in terms of the 4-bit block generate and propagate signals,

$$c_{k+4} = g_{k+3:k} + p_{k+3:k} c_k \tag{22.13}$$

Thus, the carryout from a 4-bit-wide block can be computed in only four gate delays (the first to compute p_i and g_i for $i = k - k + 3$, the second to evaluate $p_{k+3:k}$, the second and third to evaluate $g_{k+3:k}$, and the third and fourth to evaluate c_{k+4} using Equation 22.13.

An n-bit carry lookahead adder requires $\lceil (n-1)/(r-1) \rceil$ lookahead blocks, where r is the width of the block. A 4-bit lookahead block is a direct implementation of Equation 22.8 through Equation 22.12, requiring 14 logic gates. In general, an r-bit lookahead block requires $\frac{1}{2}(3r + r^2)$ logic gates. The Manchester carry chain [Kilburn et al. 1960] is an alternative switch-based technique for implementation of the lookahead block.

Figure 22.4 shows the interconnection of 16 modified full adders and 5 lookahead logic blocks to realize a 16-bit carry lookahead adder. The sequence of events that occurs during an add operation is as follows: (1) apply A, B, and carry in signals; (2) each adder computes P and G; (3) first-level lookahead logic computes the 4-bit propagate and generate signals; (4) second-level lookahead logic computes c_4, c_8, and c_{12}; (5) first-level lookahead logic computes the individual carries; and (6) each adder computes the sum outputs. This process may be extended to larger adders by subdividing the large adder into 16-bit blocks and using additional levels of carry lookahead (e.g., a 64-bit adder requires three levels).

The delay of carry lookahead adders is evaluated by recognizing that an adder with a single level of carry lookahead (for 4-bit words) has six gate delays and that each additional level of lookahead increases the maximum word size by a factor of four and adds four gate delays. More generally [Waser and Flynn 1982, pp. 83–88], the number of lookahead levels for an n-bit adder is $\lceil \log_r n \rceil$ where r is the maximum number of inputs per gate. Because an r-bit carry lookahead adder has six gate delays and there are four additional gate delays per carry lookahead level after the first,

$$\text{DELAY}_{\text{CLA}} = 2 + 4\lceil \log_r n \rceil \tag{22.14}$$

The complexity of an n-bit carry lookahead adder implemented with r-bit lookahead blocks is n modified full adders (each of which requires 8 gates) and $\lceil (n-1)/(r-1) \rceil$ lookahead logic blocks [each of which requires $\frac{1}{2}(3r + r^2)$ gates]:

$$\text{GATES}_{\text{CLA}} = 8n + \frac{1}{2}(3r + r^2)\left\lceil \frac{n-1}{r-1} \right\rceil \tag{22.15}$$

For the currently common case of $r = 4$,

$$\text{GATES}_{\text{CLA}} \approx 12\frac{2}{3}n - 4\frac{2}{3} \tag{22.16}$$

The carry lookahead approach reduces the delay of adders from increasing linearly with the word size (as for ripple carry adders) to increasing as the logarithm of the word size. As with ripple carry adders, the carry lookahead adder complexity grows linearly with the word size (for $r = 4$, this occurs at a 40% faster rate than for ripple carry adders).

22.3.1.4 Carry Skip Adder

The carry skip adder divides the words to be added into blocks (like the carry lookahead adder). The basic structure of a 16-bit carry skip adder implemented with five 3-bit blocks and one 1-bit-wide block is shown in Figure 22.5. Within each block, a ripple carry adder is used to produce the sum bits and the carry (which is used as a block generate). In addition, an AND gate is used to form the block propagate signal. These signals are combined using Equation 22.13 to produce a fast carry signal.

For example, with $k = 4$, Equation 22.13 yields $c_8 = g_{7:4} + p_{7:4}c_4$. The carry out of the second ripple carry adder is a block generate signal if it is evaluated when carries generated by the data inputs (i.e., $a_{7:4}$ and $b_{7:4}$ in Figure 22.5) are valid but before the carry that results from c_4. Normally, these two types of carries coincide in time, but in the carry skip adder the c_4 signal is produced by a 4-bit ripple carry adder, so the carry output is a block generate from 11 gate delays after application of A and B until it becomes c_8 at 19 gate delays after the application of A and B.

In the carry skip adder, the first and last blocks are simple ripple carry adders, whereas the $\lceil n/k \rceil - 2$ intermediate blocks are ripple carry adders augmented with three gates. The delay of a carry skip adder is the sum of $2k + 3$ gate delays to produce the carry in the first block, 2 gate delays through each of the intermediate blocks, and $2k + 1$ gate delays to produce the most significant sum bit in the last block.

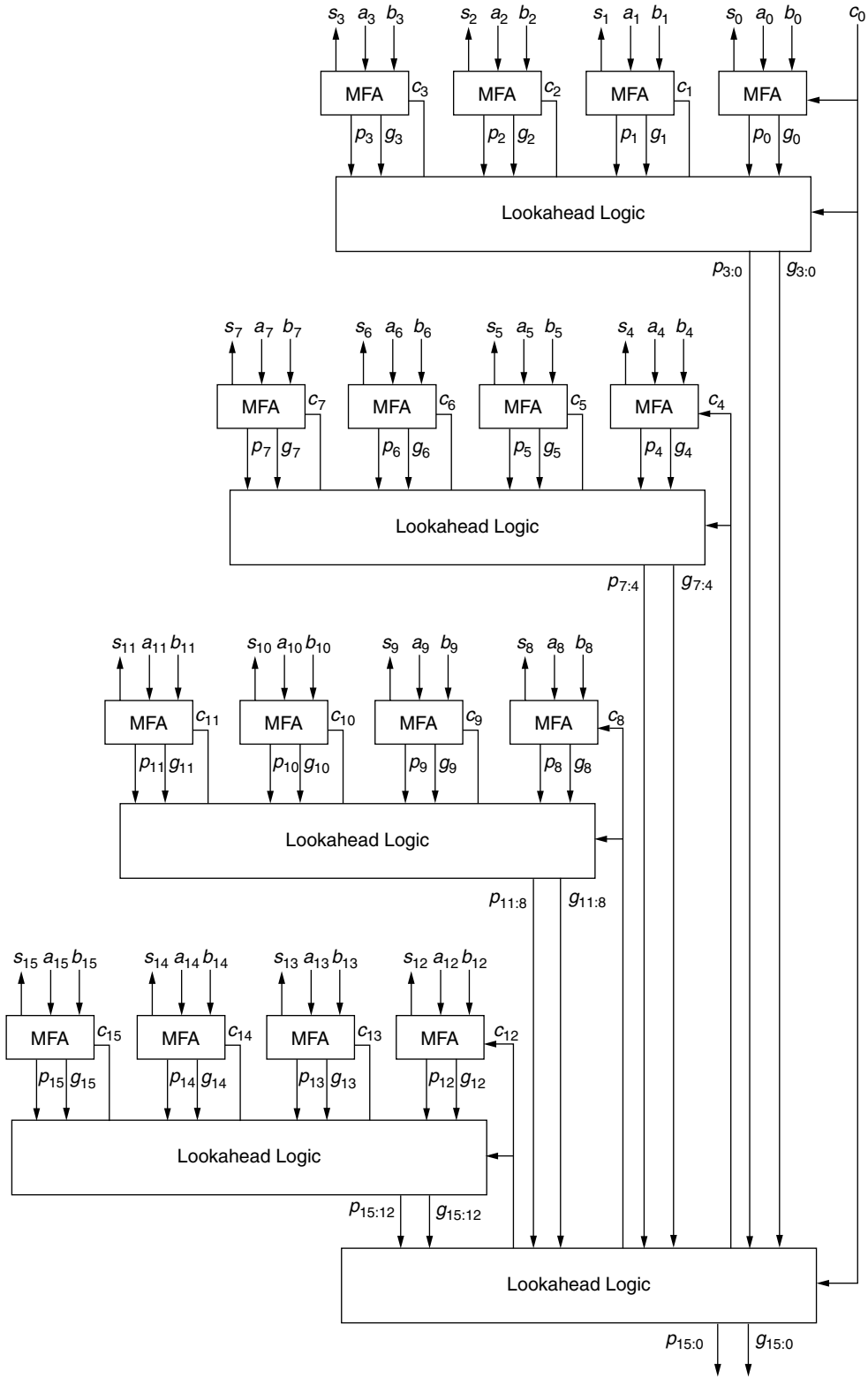

FIGURE 22.4 16-bit two level carry lookahead adder.

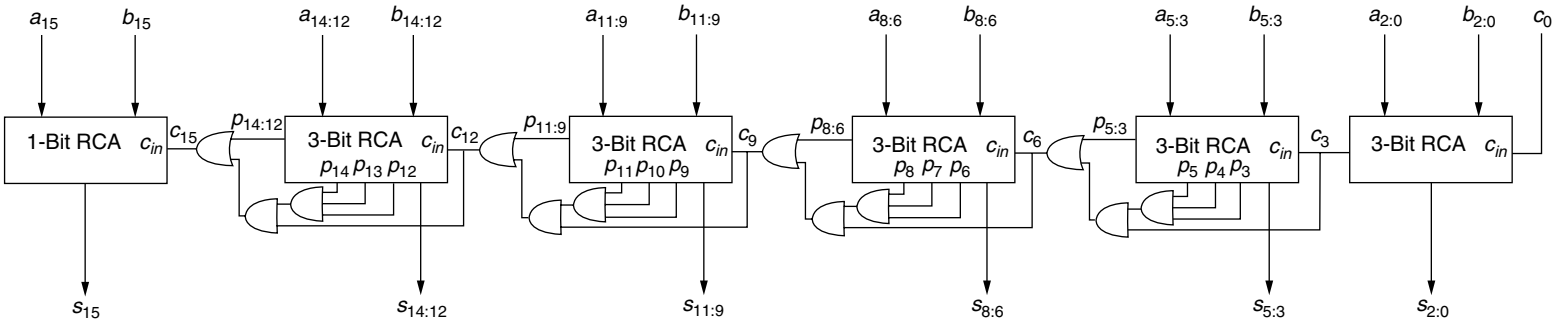

FIGURE 22.5 16-bit carry skip adder.

To simplify the analysis, the ceiling function in the count of intermediate blocks is ignored. If the block width is k,

$$\begin{aligned} \text{DELAY}_{\text{SKIP}} &= 2k + 3 + 2\left(\frac{n}{k} - 2\right) + 2k + 1 \\ &= 4k + 2\frac{n}{k} \end{aligned} \tag{22.17}$$

where $\text{DELAY}_{\text{SKIP}}$ is the total delay of the carry skip adder with a single level of k-bit-wide blocks. The optimum block size is determined by taking the derivative of $\text{DELAY}_{\text{SKIP}}$ with respect to k, setting it to zero, and solving for k. The resulting optimum values for k and $\text{DELAY}_{\text{SKIP}}$ are

$$k = \sqrt{n/2} \tag{22.18}$$

$$\text{DELAY}_{\text{SKIP}} = 4\sqrt{2n} \tag{22.19}$$

Better results can be obtained by varying the block width so that the first and last blocks are smaller while the intermediate blocks are larger and also by using multiple levels of carry skip [Chen and Schlag 1990, Turrini 1989].

The complexity of the carry skip adder is only slightly greater than that of a ripple carry adder because the first and last blocks are standard ripple carry adders while the intermediate blocks are ripple carry adders with three gates added for carry skipping.

$$\text{GATES}_{\text{SKIP}} = 9n + 3\left(\left\lceil \frac{n}{k} \right\rceil - 2\right) \tag{22.20}$$

22.3.1.5 Carry Select Adder

The carry select adder divides the words to be added into blocks and forms two sums for each block in parallel (one with a carry in of 0 and the other with a carry in of 1). As shown for a 16-bit carry select adder in Figure 22.6, the carryout from the previous block controls a multiplexer that selects the appropriate sum. The carryout is computed using Equation 22.13, because the group propagate signal is the carryout of an adder with a carry input of 1 and the group generate signal is the carryout of an adder with a carry input of 0.

If a constant block width of k is used, there will be $\lceil n/k \rceil$ blocks and the delay to generate the sum is $2k + 3$ gate delays to form the carryout of the first block, 2 gate delays for each of the $\lceil n/k \rceil - 2$ intermediate

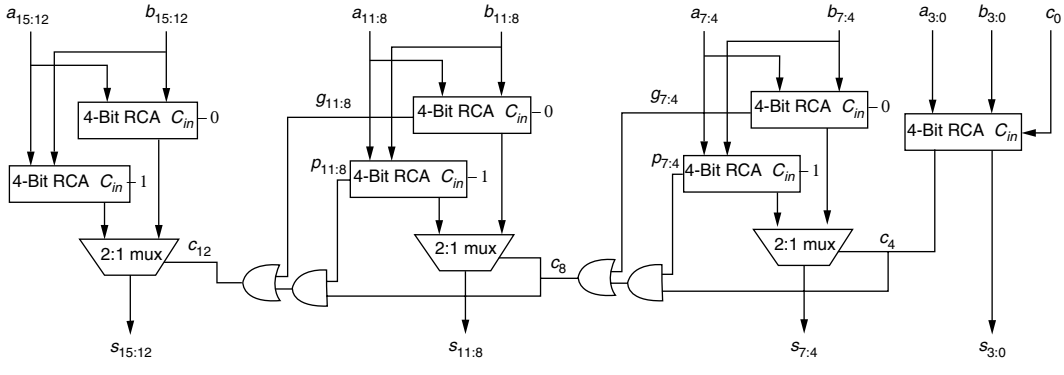

FIGURE 22.6 16-bit carry select adder.

blocks, and 3 gate delays (for the multiplexer) in the final block. To simplify the analysis, the ceiling function in the count of intermediate blocks is ignored. Thus, the total delay is

$$\text{DELAY}_{\text{C-SEL}} = 2k + 2\frac{n}{k} + 2 \tag{22.21}$$

The optimum block size is determined by taking the derivative of $\text{DELAY}_{\text{SEL}}$ with respect to k, setting it to zero, and solving for k. The results are:

$$k = \sqrt{n} \tag{22.22}$$

$$\text{DELAY}_{\text{SEL}} = 2 + 4\sqrt{n} \tag{22.23}$$

As for the carry skip adder, better results can be obtained by varying the width of the blocks. In this case, the optimum is to make the two least significant blocks the same size and each successively more significant block 1 bit larger. In this configuration, the delay for each block's most significant sum bit will equal the delay to the multiplexer control signal [Goldberg 1990, p. A-38].

The complexity of the carry select adder is $2n - k$ ripple carry adder stages, the intermediate carry logic and $(\lceil n/k \rceil - 1)$ k-bit-wide 2:1 multiplexers:

$$\begin{aligned} \text{GATES}_{\text{SEL}} &= 9(2n - k) + 2\left(\left\lceil \frac{n}{k} \right\rceil - 2\right) + 3(n - k) + \left\lceil \frac{n}{k} \right\rceil - 1 \\ &= 21n - 12k + 3\left\lceil \frac{n}{k} \right\rceil - 5 \end{aligned} \tag{22.24}$$

This is slightly more than twice the complexity of a ripple carry adder of the same size.

22.3.2 Fixed Point Subtraction

To produce an adder/subtracter, the adder is modified as shown in Figure 22.7 by including EXCLUSIVE-OR gates to complement operand B when performing subtraction. It forms either $A + B$ or $A - B$. In the case of $A + B$, the mode selector is set to logic 0, which causes the EXCLUSIVE-OR gates to pass operand B through unchanged to the ripple carry adder. The carry into the least significant adder stage is also set to ZERO, so standard addition occurs. Subtraction is implemented by setting the mode selector to logic ONE, which causes the EXCLUSIVE-OR gates to complement the bits of B; formation of the two's complement of B is completed by setting the carry into the least significant adder stage to ONE.

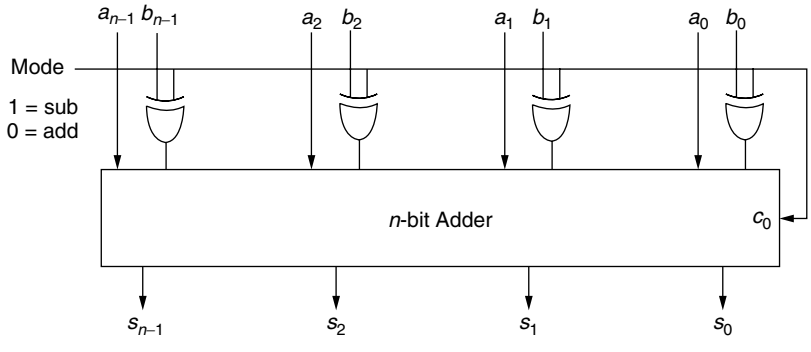

FIGURE 22.7 Adder/subtracter.

22.3.3 Fixed Point Multiplication

Multiplication is generally implemented either via a sequence of addition, subtraction, and shift operations or by direct logic implementations.

22.3.3.1 Sequential Booth Multiplier

The Booth algorithm [Booth 1951] is widely used for two's complement multiplication because it is easy to implement. Earlier two's complement multipliers (e.g., [Shaw 1950]) require data-dependent correction cycles if either operand is negative. In the Booth algorithm, to multiply A times B, the product P is initially set to 0. Then, the bits of the multiplier A are examined in pairs of adjacent bits, starting with the least significant bit (i.e., $a_0\ a_{-1}$) and assuming $a_{-1} = 0$:

- If $a_i = a_{i-1}$, $P = P/2$.
- If $a_i = 0$ and $a_{i-1} = 1$, $P = (P + B)/2$.
- If $a_i = 1$ and $a_{i-1} = 0$, $P = (P - B)/2$.

The division by 2 is not performed on the last stage (i.e., when $i = n - 1$). All of the divide by 2 operations are simple arithmetic right shifts (i.e., the word is shifted right one position and the old sign bit is repeated for the new sign bit), and overflows in the addition process are ignored. The algorithm is illustrated in Figure 22.8, in which products for all combinations of $\pm 5/8$ times $\pm 3/4$ are computed for 4-bit operands. The sequential Booth multiplier requires n cycles to form the product for a pair of n-bit

Positive Times Positive $A = \frac{5}{8} = 0.101$ $B = \frac{3}{4} = 0.110$

i	a_i	a_{i-1}	Operation	Result
0	1	0	$P = (P - B)/2$	1.1010
1	0	1	$P = (P + B)/2$	0.00110
2	1	0	$P = (P - B)/2$	1.101110
3	0	1	$P = P + B$	0.011110

Thus: $P = 0.011110 = \frac{15}{32}$

Negative Times Positive $A = -\frac{5}{8} = 1.011$ $B = \frac{3}{4} = 0.110$

i	a_i	a_{i-1}	Operation	Result
0	1	0	$P = (P - B)/2$	1.1010
1	1	1	$P = P/2$	1.11010
2	0	1	$P = (P + B)/2$	0.010010
3	1	0	$P = P - B$	1.100010

Thus: $P = 1.100010 = -\frac{15}{32}$

Positive Times Negative $A = \frac{5}{8} = 0.101$ $B = -\frac{3}{4} = 1.010$

i	a_i	a_{i-1}	Operation	Result
0	1	0	$P = (P - B)/2$	0.0110
1	0	1	$P = (P + B)/2$	1.1010
2	1	0	$P = (P - B)/2$	0.010010
3	0	1	$P = P + B$	1.100010

Thus: $P = 1.100010 = -\frac{15}{32}$

Negative Times Negative $A = -\frac{5}{8} = 1.011$ $B = -\frac{3}{4} = 1.010$

i	a_i	a_{i-1}	Operation	Result
0	1	0	$P = (P - B)/2$	0.0110
1	1	1	$P = P/2$	0.00110
2	0	1	$P = (P + B)/2$	1.101110
3	1	0	$P = P - B$	0.011110

Thus: $P = 0.011110 = \frac{15}{32}$

FIGURE 22.8 Example of sequential Booth multiplication.