Storage hierarchies

by E. I. Cohen
G. M. King
J. T. Brady

The storage hierarchy is a natural structure, given the set of available technologies and their price and performance characteristics. The physical structure of the storage subsystem is described, and the flow of data through the system is traced. The concept of a storage hierarchy is discussed, and the specific components of the IBM storage hierarchy from the processor high-speed buffer (HSB) to the on-line DASD configuration are described in detail. Trade-offs between technologies and the interactions among the levels of the hierarchy are discussed. In particular, the importance of the I/O boundary, processor storage volatility, and data sharing are highlighted. A continuous increase in virtual storage capacity can be seen in the evolution of large-scale operating systems, and MVS/ESA™ now provides the ultimate virtual capacity and function. New virtual structures available in MVS/ESA are discussed, and their relationship to the storage hierarchy is studied. The importance of storage to the performance and cost of a large processing system leads to a discussion of guidelines for storage configuration and data placement within the hierarchy.

With the continued introduction of more powerful processing systems based on faster, smaller, and more reliable technology, the requirements on storage subsystems become more and more stringent. There is also a greater focus on storage cost and performance, since storage is a key determinant of total system cost, throughput capacity, and responsiveness. The types of available technology choices allow for the configuration of a storage structure consisting of a hierarchy of storage levels, which taken together meet the diverse requirements placed on the storage subsystem. The significant new development in this hierarchy is the increasing importance of processor storage in diverting activity from the lower levels of the storage hierarchy, thus reducing system I/O rates. With the logical structure provided by Enterprise Systems Architecture/370™, increased usage of processor storage can be quick and orderly, by extending the existing concept of virtual storage and addressability. This approach offers advantages in many aspects of system performance, system structure, system cost, and system maintenance and tuning.

Before discussing the advantages and disadvantages of various storage options, one first must focus on the objectives or desired characteristics of storage. Access speed and bandwidth are primary among these objectives. High-speed access is required to respond to a storage request without delaying the central processor (CP) that is making the request. High bandwidth provides the ability to service both the sustained and peak requirements summed over all CPs.

The storage must be configurable into almost unlimited sizes to meet the data capacities required by ultra-large processing systems. Other factors such as cost, reliability, power, space, and cooling must also be reasonable. These items, however, are not areas of focus for this paper.

If one were to specify the ideal storage, one might imagine that the requirements would lead naturally to a single-level, monolithic store consisting of only one type of technology. The resulting store might be like a shared processor cache of near-infinite size.

Such a storage structure would allow full realization of a processor's potential capacity, with no penalty for storage delays. In reality there is no single technology that can provide speed, bandwidth, capacity, and low cost. There are a wide variety of storage technologies available which cover a wide range of performance. Faster technologies are usually more expensive and are more difficult to configure in bulk. Therefore, the best approach is to use the available options in combination, taking advantage of the best features of each. In this way, one can attempt to meet the full set of storage objectives as closely as possible. A detailed and far-ranging discussion of the impact of memory systems on system structure and architecture can be found in Reference 1. We expect storage technologies to improve over time, and we expect logic technologies to improve as well, both of which will produce even more stringent requirements for storage performance. Thus a hierarchical storage strategy will be applicable in the future and will probably be an even more important contributor to system performance.

Storage hierarchy data flow

The physical structure of the storage hierarchy can be viewed as seen in Figure 1, which depicts a simplified diagram of an IBM 3090 triadic system. The diagram shows three CPUs, each with an integrated processor cache which is also known as the high-speed buffer (HSB). The system control element (SCE) is the central and most complex component in this structure, because it routes data among all the CPUs, central storage, expanded storage, and all channels, while ensuring that the data elements are correct and up-to-date.

The flow of data through the physical components of the hierarchy can best be understood by the sequence of actions taken when a CPU accesses data elements resident in each storage type. The best performance case is a cache hit, because the data element can be transferred directly to the appropriate CPU component in nanoseconds. If the data element is not in the cache, it must be retrieved from the central storage through the SCE. These cache misses typically take hundreds of nanoseconds to resolve, but the specific data element requested can be routed to the CPU component in parallel with the entire line being transferred to cache.

A page fault occurs when the data element is not resident in central storage. With the advent of expanded storage technology, many page faults can now be satisfied in the tens of microseconds needed to move a 4K-byte block from the expanded store to the central store via the SCE. If the data element is not resident in either central storage or expanded storage, it must be retrieved from one of the system paging datasets via a READ I/O. In this case, the data element is transferred from the DASD through the control unit to the channel. The channel transfers the data to central storage via the SCE. At this point, the data are brought into the cache of a CPU via a cache miss the next time the requesting program is scheduled to execute.

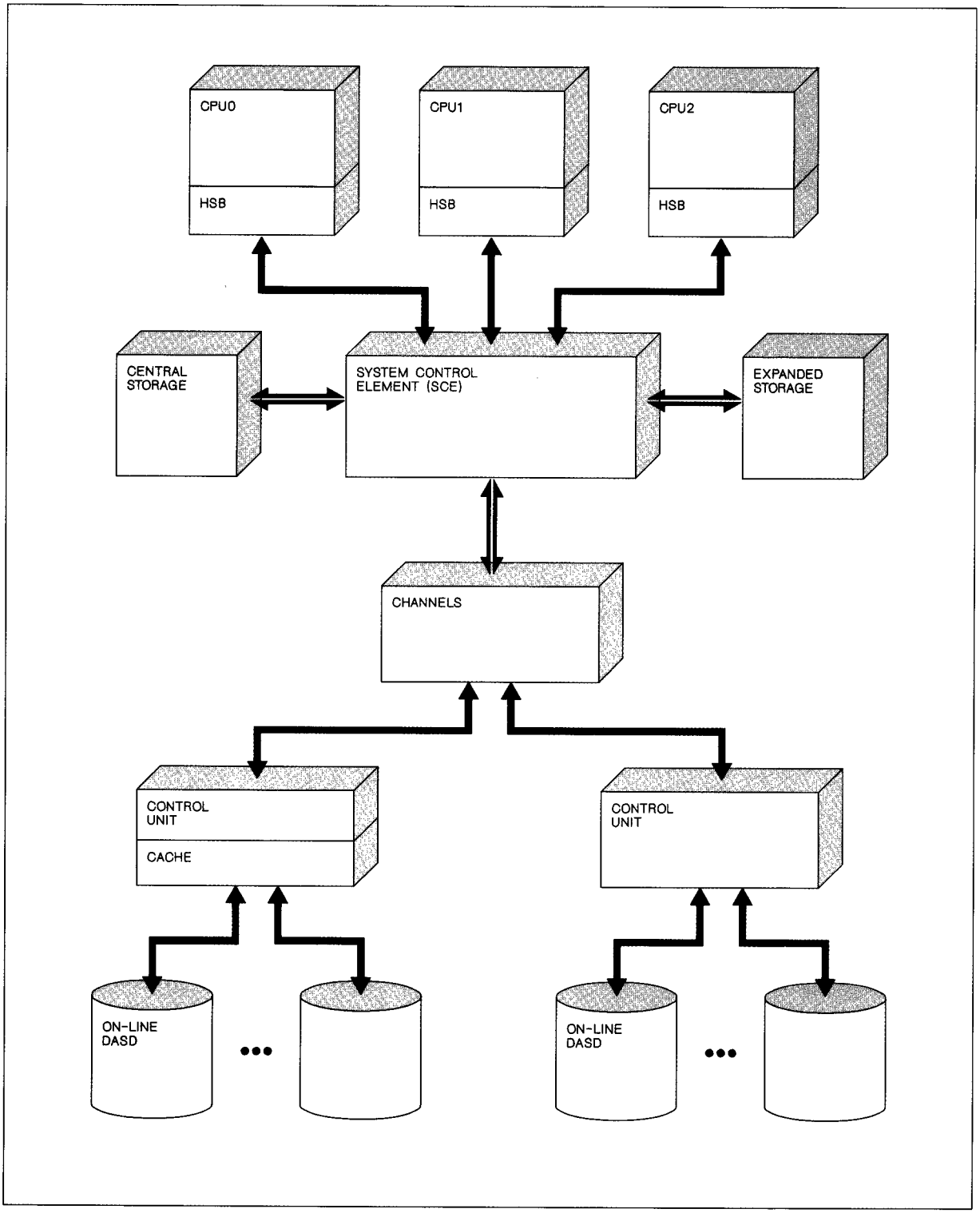
In some cases, data can be retrieved much faster if present in a control unit cache.

Data accessed from nonpaging DASD on a typical I/O request follow the same path as that of the paging case previously described. In either case, the I/O can take tens of milliseconds if the data need to be retrieved from the DASD. In some cases, data can be retrieved much faster if present in a control unit cache. In this case, the physical DASD need not be accessed, and the requested data can be transferred directly from the electronic storage in the control unit in about 2–3 milliseconds. This is logically equivalent to finding the data one level higher in the hierarchy on a faster but smaller capacity technology. This physical hierarchy can vary depending on machine type and system configuration, but when viewed conceptually we can study its general features.

Elements of the IBM storage hierarchy

In designing a storage system consisting of multiple heterogeneous technologies, one tries to take advantage of the key values of each technology, thus meeting all desired characteristics. This storage system structure is conceptually viewed as a pyramid of levels as in Figure 2. The levels are numbered se-

Figure 1 Storage hierarchy data flow



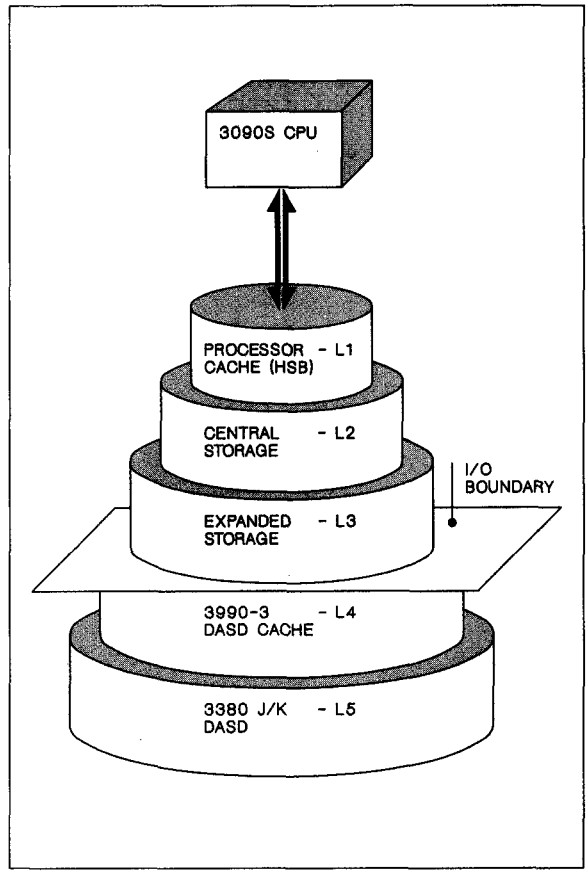
quentially starting from level 1 (L1), as we descend the hierarchy. The CPUs are positioned above the top level of the hierarchy because data elements are brought to the top level of the hierarchy for the CPU to access them.

The key characteristics in understanding the concept of a storage hierarchy are access speed, capacity, and cost. As we move downward through the hierarchy, each subsequent level is slower and less expensive and usually configured in larger cost-effective capacities. The overall objective of the storage hierarchy is to provide average access speed almost as fast as the fastest level (HSB) with an average cost per bit of on-line data almost as low as the least expensive level (DASD). This can be achieved only if the vast majority of on-line data are resident on DASD while almost all of the CPUs storage accesses are satisfied from high-speed buffers.

Data elements are moved upward and downward through the storage hierarchy based on reference activity. Storage reference patterns show a strong locality of reference, which produces a high degree of data reuse and allows a very high hit ratio in the HSB with a very small working set. Each level of the hierarchy is managed by a system component (hardware, software, or microcode). The management approach is generally based on a least recently used (LRU) algorithm for replacing an inactive data element by a more active element. The element replaced is moved downward in the hierarchy, while the requested element is rising to the top of the hierarchy.

Given its primary position in the hierarchy, the performance of the HSB has a significant impact on the speed of the processor and the overall system performance. Levels 2 and 3 together constitute the processor storage (PS). Central storage (CS) is the byte-addressable main storage of the processor system, and expanded storage (ES) is the 4K-byte block addressable extension to the central storage. The DASD subsystem is made up of DASD and DASD caches. As the home location of the bulk of permanent on-line data, the DASD subsystem fulfills the role of the cost-effective bulk storage in the hierarchy. The DASD cache can be very effective in reducing the average response time and in reducing the access rate to the DASD. Tape could be added as level 6, but this is not normally a point of focus because of its archival rather than on-line nature. Each of these hierarchical elements is discussed in further detail in the following sections.

Figure 2 IBM storage hierarchy elements



Processor high-speed buffer

The top level of the storage hierarchy is the processor cache, which is also called the high-speed buffer (HSB). It is the most critical element in the hierarchy, because it satisfies processor storage references (both instructions and data) directly to the CPU logic elements and registers. The HSB performance, therefore, has a direct and powerful impact on processor speed and system capacity. The processor cache needs to operate in the same range of speed as the processor to satisfy its storage references without causing any CPU delays. This speed requirement also means that the cache must be physically close to the processor logic to minimize propagation delays. The L1 cache is typically tens to low hundreds of kilobytes in size, because of the high cost and spatial limitations. Processor caches in this size range are extremely effective, because of the strong locality of reference

exhibited by a program's instruction and data reference sequences and typically can satisfy 95 to more than 99 percent of all storage requests. When a referenced data element is not found in the cache, it must be retrieved from central storage. This is a much slower access, and the processor may become idle while waiting for a data element to be delivered. In this case, the requested data element is routed to both the processor logic and the cache in parallel, to minimize this delay.

The cache is transparent to the program executing and is managed by the hardware logic in the buffer control element.

The cache is transparent to the program executing on the processor and is completely managed by the hardware logic in the buffer control element. This is done totally independently of the software, and the software has no knowledge of whether the data requested are coming from the cache or the central storage. In fact, the software is unaware of whether the processor has a cache, and the same software can operate on a processor without a cache, albeit more slowly. The cache contents are managed within the processor hardware using an LRU-based replacement algorithm that is similar in philosophy to other memory managers implemented in software or microcode in other parts of the storage hierarchy.

The effectiveness of a processor cache is usually described by a cache (or buffer) hit ratio, which is calculated as the percent of total storage references satisfied directly from the buffer with no access to central storage. Although the typical buffer hit range seems to provide high buffer performance, improvements within the range are very important to CPU performance. Although there seems to be little difference between 96 and 98 percent buffer hit ratios, the impact is pronounced because the buffer miss ratio drops from 4 to 2 percent. This means that half as many accesses to central storage are required and the overall CPU delay due to storage is cut in half. Looking at it from the viewpoint of cache misses gives a truer picture of the impact of cache performance on total processor performance.

There are two distinct management philosophies for processor caches based on when updates to data are pushed through to central storage. In a *store-through design*, updates are made to the cache and to the corresponding data in central storage, the home location of the data. In a *store-in cache*, updates are made only to cache; these updates are reflected back to central storage only when the data element is removed from the cache by LRU replacement. The advantage of the store-through design is its simplicity. All updates are made to central storage immediately, which makes the newly changed data available to other processors from its central storage home location. A store-in cache has the advantage of reducing transfers between cache and central storage, because changes are written back to central storage only when they are absolutely required. The controls required for store-in are more complex than store-through, particularly in tightly coupled CPUs. In this case, a storage reference on one processor may need to be checked against the contents of caches on other CPUs to determine whether it contains an updated version of the data. Overall, one cannot say whether one approach is superior, because much depends on the objectives of the design (e.g., the trade-off between efficient use of bandwidth and simplicity of design). Either design may be the best solution, depending on the set of design criteria.

Processor storage

Processor storage includes the *synchronous levels* of the storage hierarchy. That is, the processor waits while an item of data is retrieved from a level of this class of storage. Processor storage is closely integrated with the processor, and the time to retrieve an item from it is measured in microseconds or less. Because this access time is so fast, it is more efficient for the processor to pause momentarily to wait for an item to be retrieved from processor storage than to have the processor search for another task to execute. (See the section on the importance of the I/O boundary later in this paper.)

The size of processor storage generally ranges from tens of megabytes to several gigabytes and consists of multiple 4096-byte blocks, called *frames* of processor storage. For example, a configuration of 256 megabytes of processor storage contains 65 536 frames. The contents of processor storage are man-

aged by the operating system. Through the use of a global LRU scheme, the operating system tends to keep the most actively referenced programs and data areas of the users of the system in processor storage. There are two types of processor storage in the IBM storage hierarchy: *central storage* and *expanded storage*, the characteristics of which are discussed in the following sections.

Central storage. Central storage (which is often called real storage) is byte-addressable processor storage. That is, an address generated for central storage points to one byte of central storage. With the System/370-XA and ESA/370™ architectures, 31-bit addresses are defined for central storage. Thus the maximum size central storage allowed by these architectures is 2 gigabytes. For each frame of central storage, the hardware maintains reference, change, and access key indicators in special high-speed storage packaged close to the processors. These indicators are used by the operating system to preserve data integrity and to assist in its LRU management of central storage. Data reliability is maintained through the hardware use of single-error-correcting, double-error-detecting codes packaged with the central storage.

Although the buffer hit range seems to provide high performance, improvements within the range are very important.

Although conceptually sitting between two levels of the storage hierarchy, central storage is physically accessable to three levels: the high-speed buffer (HSB) of each processor, expanded storage, and the DASD subsystem. When a processor requires a data item that is located in central storage, the relatively small piece of central storage that contains the item is moved into its HSB. Such a piece of central storage is 128 bytes on an IBM 3090S processor. This operation is managed by the hardware and takes a fraction of a microsecond to complete. See the section on processor high-speed buffers presented earlier in this paper. Transfers between central storage and expanded storage are accomplished in frame-size pieces under the control of the operating system in times measured in tens of microseconds. This is discussed in greater detail in the section that follows. Applications usually originate requests for data movement between central storage and the DASD subsystem. These moves vary in size typically from 80 to 32 768 bytes or more and are made through a channel interface in times measured in milliseconds. This is discussed later in this paper under the subject of DASD.

The operating system keeps the most active set of programs and data areas in central storage. Generally, items can remain unreferenced in central storage for tens of seconds before they are moved out to a lower level of the hierarchy.

Expanded storage. Expanded storage is block-addressable processor storage. An address generated for expanded storage points to a 4096-byte block or frame of expanded storage. A 32-bit address is defined for expanded storage, thereby providing addressability for up to 16 terabytes (that is, 16 384 gigabytes) of expanded storage. The operating system maintains information in a control block for each frame of expanded storage, and that information is used for preserving data integrity and for LRU management of expanded storage. Unlike central storage, there is no need for special circuitry to provide these functions. Data reliability in expanded storage is enhanced through the use of double-error-correcting, triple-error-detecting codes packaged with the expanded storage.

Conceptually, expanded storage sits between the central storage and the DASD subsystem levels of the storage hierarchy. However, expanded storage is physically accessable only to central storage. When a processor requires a data item that is located in expanded storage, the contents of the expanded storage frame containing the item are moved into a central storage frame. This operation is managed by the operating system and takes tens of microseconds to complete. For example, on a 3090S processor, this movement takes approximately 75 microseconds, including the hardware and operating system components. Once the data resides in central storage, it is then moved into the processor's HSB, as described in the preceding section on central storage. Data

movement to and from expanded storage causes little interference to the other processors in a multiprocessor system. Transfers between central storage and the other processor HSBs do not wait for an expanded storage transfer to complete. However, should more than one processor request access to expanded storage, those requests are usually satisfied sequentially.

The operating system keeps active programs and data areas in expanded storage. This is in addition to the most active programs and data areas, which are kept in central storage. A global LRU algorithm is used across the combination of central and expanded storage. The most active data are placed in central storage, the next most active data reside in expanded storage. Generally, items can remain unreferenced in expanded storage for hundreds or even thousands of seconds before they are moved out to a lower level of the hierarchy. The operating system accomplishes the movement to the level below expanded storage by first moving the data into central storage and then sending it to the DASD subsystem.

DASD

In the large systems of IBM, the base of the active storage hierarchy is the direct-access storage device (DASD). These devices are built on the magnetic disk technology with mechanical actuators, and they represent a highly evolved trade-off between storage cost and device performance. Typical DASD configurations provide for tens to hundreds of gigabytes of storage.

Response time. The most critical measure of DASD performance is response time, which is composed of two components: service time and queuing time. *Service time* is defined to be the sum of the following times: seek, latency, RPS miss, and data transfer. Service time is a raw measure of the performance of the DASD.

The time to move the actuator from one location on the disk to another is called *seek time.* The IBM 3380 Model K has a rated average seek time of 16 milliseconds. However, due to the reference pattern to the data, in most cases the experienced average seek is about 25 to 30 percent of the rated average seek.

Latency is the time delay associated with the rotation of the disc storage medium until the requested data field is located under the read/write head. Generally, latency is stated in terms of the time it takes to complete a half revolution of the disc. The IBM 3380 and many other DASD devices rotate at about 3600 RPM, giving a latency of 8.3 milliseconds.

Once the proper record on the disk has rotated under the read/write head, the device is ready to transmit data back to the channel. If the channel is busy servicing another device, the opportunity to transmit data is missed and a full rotation is required before the read/write head is properly positioned over the record again. This additional delay is called a *rotational position sensing* (RPS) *miss.* The number of misses that can be seen by a given I/O operation is a function of the utilization of the channel, control unit, and pathing configuration.

Minimization of DASD response time is a primary objective in the design of a storage hierarchy.

Data transfer time is the time it takes to move the data from the device to the central storage of the processor. It may be calculated by dividing the number of bytes to be transferred by the transfer rate of the DASD. The average record size for most on-line and interactive environments ranges from 4000 to 8000 bytes, with a modest upward trend over time. The IBM 3380 DASD devices transfer at 3 million bytes per second. This yields average transfer times in the range of 1 to 3 milliseconds. With a seek time in the the 4 to 5 millisecond range and latency above 8 milliseconds, data transfer is often not a significant contributor to DASD performance. However, in batch, logging, and dump/restore applications, and cached-DASD subsystems, the seek and latency can be minimized, and the transfer rate becomes dominant.

Queuing time reflects the delay in initiating an I/O request, because the path to the device or the device is busy with another request.

Because of its effect on the overall system throughput and end-user response time, minimization of DASD

response time is a primary objective in the design of a storage hierarchy. Response time is minimized by addressing each of its components. In the current DASD and control units, one of the more effective innovations is the use of alternate pathing to reduce path contention to a point that almost eliminates the RPS miss. In these systems the electronics provide up to four alternate paths from the device to central storage.

Long-term trends in processor and DASD technology show a 10 percent compound increase of the processor and DASD-performance gap. Significant contributors to DASD performance are based on mechanical rather than electronic technologies. Therefore, other avenues must be explored to keep pace with the DASD response time requirements of systems. Data reference patterns, for one thing, lend themselves to incorporating a more intelligent management algorithm into the electronic storage technology used in a DASD control unit.

DASD control unit cache performance. The buffering of data in a high-speed, lookaside cache depends on a characteristic of data called locality of reference. In the case of sequential access to data, there is a high probability that once a record is referenced, the succeeding record will also be accessed. In the case of nonsequential access to data, there is a high probability that having once referenced an item of data, that item will shortly be referenced again and data near it will be referenced. If a request can be serviced from the cache, seek and latency times can be avoided, thus overcoming the two most severe performance problems in the DASD subsystem.

The IBM 3880 and 3990 cache control units adopt multiple strategies to take advantage of the locality of reference. Least recently used (LRU) algorithms address the reuse of data. An LRU algorithm is used to decide what data must be destaged or deleted from a cache when all the buffers in cache have been committed. The use of LRU and the implementation of the fast write feature can allow the IBM 3990 to achieve up to 80 percent hit ratios on small-record databases. These are the most difficult databases to cache because of the smallness of the records and the random-access patterns of accessing them. In such systems, often 15 percent of the activity is WRITE I/O. The fast write capability provides a nonvolatile storage feature (NVS) in the controller that allows write hits to enjoy the same performance advantage granted to read hits to the cache. That is, seek and latency times are avoided.

Partial track staging provides two improvements. By staging the record referenced and the rest of the track following it, the cache anticipates sequential references and the nearby references associated with random databases. With a little assistance from the LRU management of cache, this allows hit ratios in the high 90 percent range to be achieved, with reasonable cache sizes. A more detailed discussion of cache-DASD performance can be found in Reference 2.

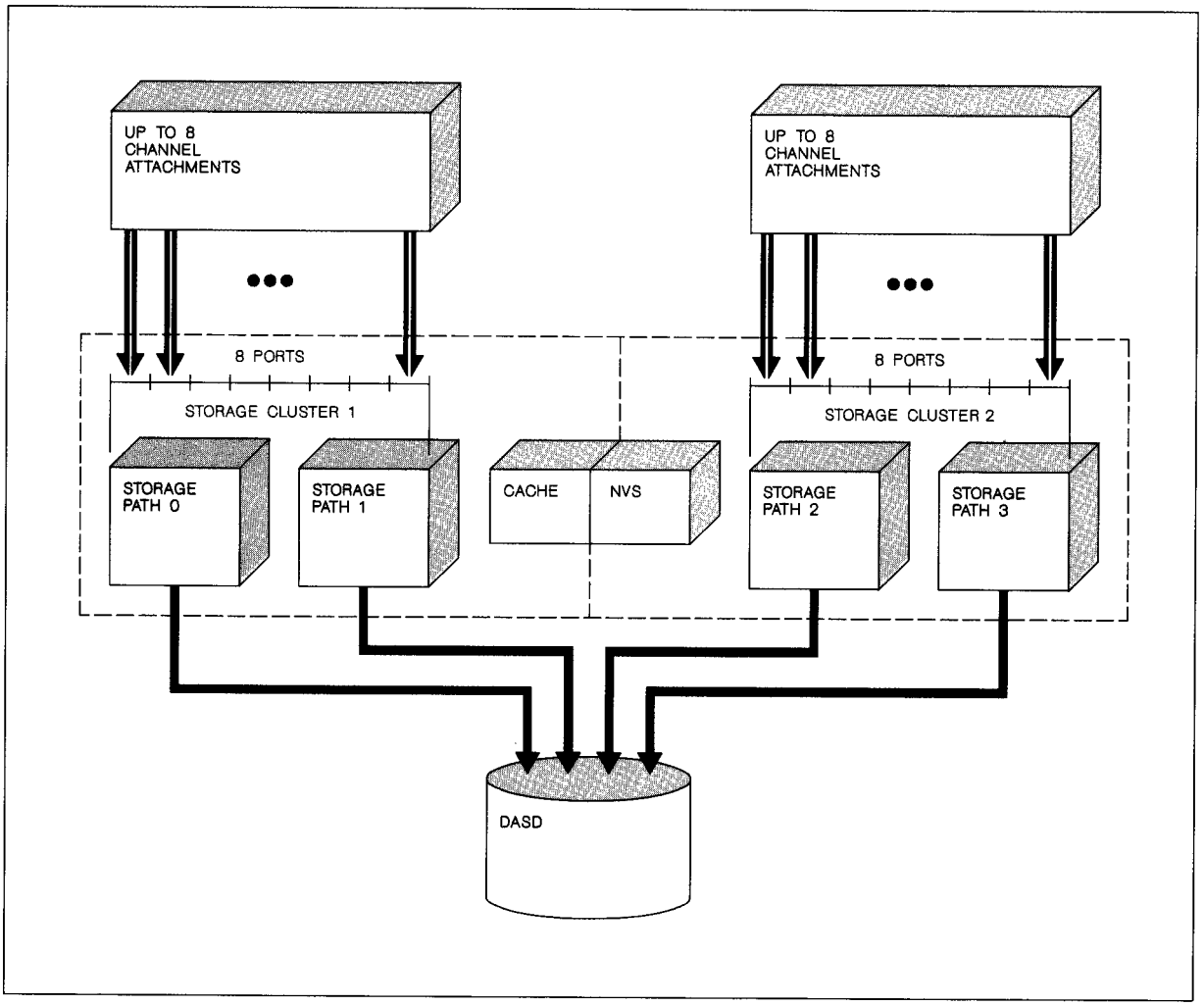
Over time, DASD has become one of the most reliable components of a computing system.

Together these features give a DASD cache controller up to an order of magnitude better service time and response time than an equivalent configuration of uncached DASD. Our current estimate is these structures and refinements on them will allow the DASD subsystem to meet the performance needs until the mid-1990s and perhaps beyond.

DASD control unit cache availability. Over time, DASD has become one of the most reliable components of a computing system. As a result, most recovery and availability software has made an assumption about the safety of data written to DASD. Whenever this assumption is wrong, the recovery is long, tedious, and costly. Thus the design of a cache control unit must offer availability and reliability characteristics very similar to those of the DASD.

The IBM 3990 control unit structure provides a number of features that enhance availability as well as performance. Each 3990 provides one or two storage directors (SDs), each of which provides two storage paths (SPs). Two SPs form a cluster and share a common power supply. A cluster has attachment capability for eight channels. One control unit can attach 16 channels, eight to each cluster. This configuration is shown in Figure 3.

Figure 3 Pathing overview



The IBM 3380 Models J and K DASD have four paths out of each actuator that can be dynamically switched to any of four SPs. Earlier DASD models had two paths from each actuator. These features are called device level selection—enhanced (DLSE) and device level selection (DLS), respectively. The channel attachment and DLSE features make it very unlikely that a path failure will result in the unavailability of data. In a configuration designed to tolerate at least one failure, it takes two channel failures, or four storage path failures, or two cluster failures, or one actuator or head disk assembly (HDA) to make data unavailable. Although the likelihood of such events is very small, a dual-copy feature permits a defense against these kinds of failures.

The NVS feature allows write I/Os to be buffered in the controller. By using the cross-connected NVS features, the system can ensure that the failure of one NVS feature is recoverable in the DASD subsystem without requiring any actions by software or operators. The dual-copy feature extends that protection to the DASD. Dual copy allows volumes to be backed up on another device and dynamically maintains an exact copy of the volumes. In the event of a failure in the HDA or actuator, the control unit automatically switches to the backup volume and notifies the system of the failure. The operations staff can then schedule service of the failed device. After repair, the operations staff informs the DASD control unit to re-establish the dual copy. Over a period of time, the

good DASD is copied to the repaired device while merging the updates made while the copy process is in progress.

Importance of the I/O boundary

The most distinct break between levels of the storage hierarchy occurs between the processor storage and the DASD subsystem levels. This point is referred to as the *I/O boundary* and denotes the place where data access switches from being synchronous with the processor to being asynchronous.

During the execution of a task, a processor continually requests the next instruction or data item associated with the task, until the task is completed. The expected time to retrieve the requested item determines whether it is more efficient for the processor to wait for the requested item to arrive than to look for a different task to execute. The break-even point for this decision occurs when the time a processor sits idle waiting for an item to arrive equals the time a processor is busy switching among the tasks being executed. A retrieval is called *synchronous* when the processor waits for the item to arrive; a retrieval is called *asynchronous* when the processor looks for something else to do instead of waiting.

An asynchronous retrieval is a disruptive event for both the hardware and the operating system. The operating system must take the following actions: set up for and schedule the retrieval; save the state of the original task; and search for a new task to dispatch. During the early stages of execution of a new task, the processor runs slowly, as it waits for its HSB to be filled with data associated with the new task by displacing the HSB data of the original task. At some point in the future, the retrieval will be completed, which leads to a switch back to the original task and a repeat of the steps just described.

On an IBM 3090S processor, the processor time overhead associated with an asynchronous retrieval is of the order of several hundred microseconds. This number includes the effects of two task switches and of the HSB contents thrashing, which might be eliminated in a synchronous retrieval. Therefore, in deciding which levels of the storage hierarchy on a 3090S to access synchronously and which to access asynchronously, the time to retrieve a data item from a hierarchy level is compared with a break-even time of several hundred microseconds. The access time to the lowest level of processor storage, expanded storage, is approximately 75 microseconds on a 3090S. (See the earlier section on expanded storage.) The highest level of the DASD subsystem, the DASD control unit cache, is capable of delivering data in about 3 milliseconds. (See the earlier sections on the DASD control unit cache.) Thus, levels of the storage hierarchy from expanded storage and above are accessed synchronously, levels from the DASD control unit cache and below are accessed asynchronously.

The most common implementation of an asynchronous retrieval is an I/O operation.

The most common implementation of an asynchronous retrieval is an I/O operation. To retrieve a data item in this manner, an application (or possibly the operating system) builds a request for an I/O operation. The operating system schedules the I/O and then performs a task switch. When the data are available, an I/O interrupt is processed by the operating system followed by an eventual task switch back to the original task. This entire process may consume 400 to 800 microseconds of processor time on a 3090S, now adding the application I/O-interface-build cost to the asynchronous retrieval overhead. Thus, the I/O boundary, that point at which access to levels of the storage hierarchy switches from synchronous to asynchronous, is significant relative to the processor-time cost of data access.

Processor storage volatility

Volatility of storage means that the contents are lost when the power is shut off. Processor storage (both central and expanded) is volatile, and the analogous element in previous systems (real or main storage) has also been volatile. Therefore, all software has been designed so as to take this volatility into account. For example, I/O buffer managers and database systems force updates out of the buffer to the DASD subsystem before the update is complete. The software has been designed to function properly with volatile processor storage and buffer pools of any size.

In a typical interactive database environment every update must be written through to nonvolatile storage in the DASD subsystem. Therefore, in this case, the use of storage for caching is not effective in reducing the number of write I/Os. Thus, the read-write ratio is an important parameter in determining the potential impact of large buffer pools, because only the read I/Os can readily be avoided. The read-write ratio in a large database system is typically of the order of five to one. Because most of the I/Os are read operations the buffering can be very effective in reducing the number of I/Os. Even in the case of a one-to-one ratio, buffering can address half of the I/Os and still make a significant reduction in the I/O rate. Additionally, even in the case of a sequential, write-only, large dataset, buffering (either by using larger blocks or chaining more blocks together per channel program) can significantly reduce the number of write I/Os needed to transfer a given amount of data.

Data sharing is a key part of providing operational flexibility, capacity, and availability.

Multisystem shared data

Data sharing is a key part of providing operational flexibility, capacity, and availability. This sharing can occur at any level of the hierarchy. The system architect must know the performance criteria for the data and its sharing characteristics to select the most efficient level.

Data that has stringent response time demands and has read-only or near-read-only characteristics can be cached in the DASD control unit or expanded storage. Expanded storage gives better performance in these cases, because there are no (few) WRITEs that must be reflected back to DASD. This limits the overhead of synchronizing multiple copies of the data. A penalty for sharing data in a processor local cache, such as expanded storage, is the cost of the storage in each local cache. If multiple processors—four for example—share a program library that is frequently used, the active programs would be included in each local cache. Experience indicates that the performance advantage of doing this by far exceeds the cost of the storage necessary to keep the three additional copies.

The cost of synchronizing storage levels must be separately assessed. As the WRITE content of a shared dataset increases, the cost of managing multiple concurrent buffers in the form of software path length and communications delays begins to mount. These performance effects eventually overshadow the advantage of expanded storage, and the DASD cache begins to emerge as having the cost/performance advantage. The READ-to-WRITE ratio, the rereference and local reference components of the hit ratio, the efficiency of the buffer management process, and the performance of the DASD all have substantial impact on the overall trade-off between a DASD cache and an expanded storage cache. In general, unshared data and shared data with a low update rate are able to cache well in a processor local cache, giving throughput and response-time advantages over a DASD cache. In the case of the shared data with a high update content, DASD cache currently has the advantage. It requires less storage to achieve the required hit ratios, to provide more consistent performance as the number of systems sharing the data increases, and to provide better availability characteristics.

Over time, the challenge to the large-system architects is to try to combine the advantages of each form of caching. As the cost of storage technology falls, channel data rates improve, and new synchronization techniques are developed, the relationship of the data cache to its backing DASD and the processor storage are expected to change and give rise to new system structures.

The impact of virtual addressability

The performance advantages of I/O avoidance are achieved through the improved use of the physical storage hierarchy through the extended use of virtual storage. In this way, the details of the physical storage can be hidden from application programs, but the value can be obtained through use of an existing facility. The use of virtual storage as the mechanism to address the storage hierarchy provides an easy-to-use system function and structure.

The permanent home location of on-line data is on DASD. Virtual data is the view from a program, and it is mapped to a subset of the hierarchy consisting of central storage, expanded storage, and auxiliary storage (i.e., the paging/swapping datasets on DASD). The approach is to capture more of the active permanent data in the virtualized hierarchy where the performance value of central and expanded storage can be realized. The increased use of virtual storage is the key to the increased use of processor storage, and the resulting reduction in read I/Os to permanent on-line data.

The major trade-off is the cost of increasing the size of a level versus the savings from reduced data movement.

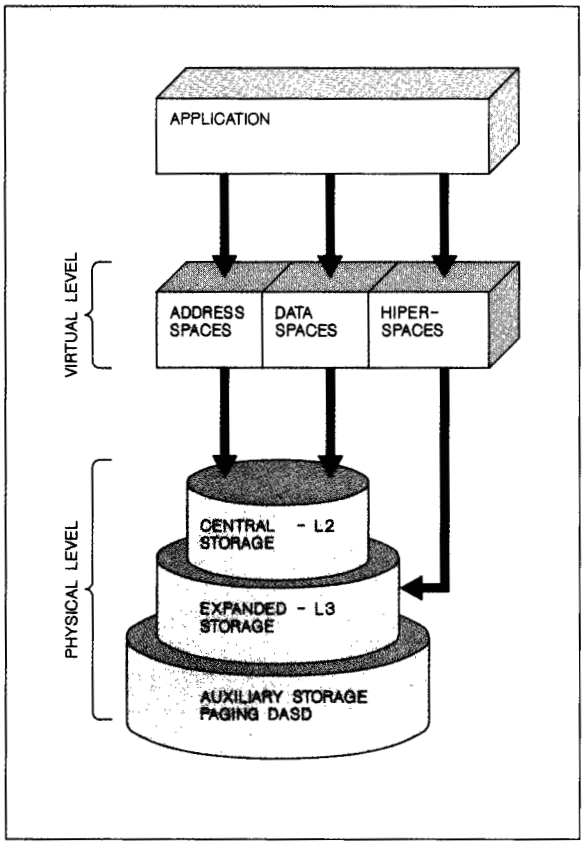
The history of storage addressing is one of continuous expansion of the maximum addressable storage available to a program. This is being driven by larger space requirements for larger, more complex programs, more system data and code, and increased buffering of user data. In the MVT operating system, which was dominant in the late 1960s and early 1970s, a single 16-megabyte real storage was shared among programs. MVT was superseded in the 1970s by MVS/370, which provided a dedicated 16-megabyte address space per program. The next dominant operating system was MVS/XA™ which was introduced in 1983 and provided a dedicated 2-gigabyte address space to each program. Although 2 gigabytes are thought to be sufficient for long-term code growth, data growth could surpass it rather quickly. A new extensible structure is required to meet the potentially explosive growth in virtualized permanent data.

This requirement can be satisfied by the virtual extensions provided by Enterprise Systems Architecture/370. In MVS/ESA, a program can gain access to one or more data spaces in addition to the address space holding the program itself. Each data space can hold 2 gigabytes of data, and a large number of data spaces can be accessible from a single address space. This type of horizontal growth—that is, 2 gigabytes for each data space added—can satisfy long-term growth in virtualizing active permanent data. In order to make data spaces immediately accessible and valuable, standard MVS functions can now use data spaces through a common system component, the virtual lookaside facility (VLF). Current application programs use data spaces indirectly by using functions which in turn use VLF. The overall function provided is a data caching function, where VLF uses data spaces to hold the cached data. With sufficient configurations of expanded storage, this cached data reside physically in expanded storage, a much higher-performance medium than permanent DASD. Many functions already use data spaces, and other functions are to be added as new system facilities become available.

The residency of virtualized data in central storage, expanded storage, and auxiliary storage is managed by demand fetch into central storage with LRU replacement from central storage to expanded storage and from expanded storage to auxiliary storage. Although LRU management is the best one can do with no special knowledge of upcoming reference patterns, at times a subsystem or application may know more about the upcoming data than does the operating system. For example, data no longer needed can be immediately moved down in the hierarchy rather than aging out due to inactivity and LRU replacement.

A subsystem or application may know the data area to be used next by the program, in which case that data can be brought into central storage via an anticipatory prefetch rather than page-by-page demand fetch. This potential is realized through the use of a hiperspace. A hiperspace is a space consisting of 4K-byte blocks of data that can be moved directly to and from the hiperspace via command rather than page by page, as its inactivity is recognized. Standard hiperspaces generally exist on expanded storage, and older pages of a hiperspace may be migrated to auxiliary storage when expanded storage is full. The definition of a hiperspace allows a subsystem or application to move blocks of data to central storage, use them, and then move them back to the hiperspace on expanded storage. Hiperspaces provide a complementary facility to address spaces and data spaces, and among them the virtual addressability can be increased easily and efficiently. In this way, ESA/370 provides the virtual extensibility that allows for the exploitation of virtual storage and through it the value of the storage hierarchy, as shown in Figure 4.

Figure 4 Virtual storage and the storage hierarchy



Storage estimates and hierarchy configuration

The determination of the size of each level of the storage hierarchy to be configured to a system involves the evaluation of several factors. The major trade-off to be analyzed is the cost of increasing the size of a level of the hierarchy versus the savings from the resulting reduced data movement rate to a lower level of the hierarchy. However, a level of the hierarchy may have unique characteristics or provide special functions that must be considered in addition to the aforementioned trade-off.

The analysis required to size two of the levels of the storage hierarchy is quite straightforward. By definition, the lowest level of the hierarchy has no lower level to which a data movement rate may be reduced. Thus the lowest level, high-capacity DASD must be large enough to contain all data. On the other hand, the highest level of the hierarchy—the processor high-speed buffer—is of fixed size, and no variable-size trade-off can be made.

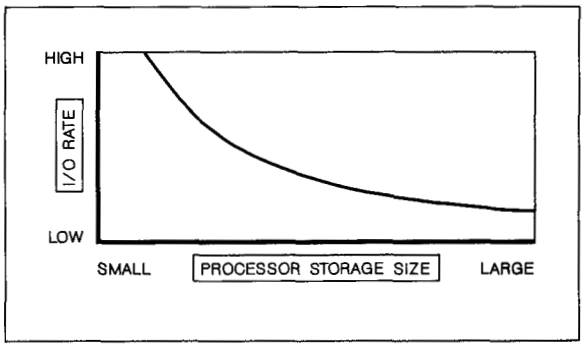
Another level of the storage hierarchy—DASD control unit cache—offers several attractive options the values of which go beyond simply reducing the data movement rate to a lower level of the hierarchy. The dual copy function provides increased availability for critical data. The DASD fast write function can provide improved performance for data that must be written to a nonvolatile medium, as discussed previously in the section on the DASD control unit cache. Both these functions are intended to *enhance* the movement of data to a lower level of the hierarchy. In any case, the dual-copy and fast-write functions generally do not reduce the data movement rate to a lower hierarchical level. The DASD control unit cache can also reduce the data retrieval (READs) from the level below it. Thus the trade-off of size versus reduced data movement rate also applies.

The sole purpose of the processor storage level of the hierarchy is to reduce data accesses to the levels below it, that is, to reduce data movement across the I/O boundary. (See the section on the importance of the I/O boundary.) A curve, such as given in Figure 5, may be drawn to illustrate the relationship between the size of processor storage and the I/O rate of a system. The shape of the curve depends on the type of workload and the extent to which data and programs have been virtualized (thus permitting their residence in processor storage). The benefits of a reduced I/O rate are numerous. As I/O is reduced, the capacity of the processor to execute transactions may be increased in two ways: (1) The processor time per transaction is reduced (as discussed in the section on the importance of the I/O boundary); (2) the processor may be able to achieve a higher utilization. A reduction in I/O generally provides significant improvements in response time, thereby increasing the productivity of the user community. Often, less system programmer time is required for tuning a system as the I/O rate drops. There may also be a savings in the I/O configuration, as the DASD access density is decreased. With these benefits in mind, the amount of processor storage to configure to a system should be chosen from the flat part of the curve. That is, choose an amount where an additional increment of processor storage would have little effect on reducing the I/O rate of the system. Reference 3 provides a methodology for estimating processor storage configurations.

The processor storage level of the storage hierarchy is actually composed of two levels: central storage and expanded storage. The choice of a central storage size determines the rate of data movement between

the two levels. A curve similar to that discussed above may be drawn, this time to illustrate the relationship between the size of central storage and the data movement rate to expanded storage. A reduction in the data movement results in an increase in processor capacity resulting from the decrease in processor time spent on page movement. Generally, no noticeable changes in response time or tuning occur from reduced page movement to expanded storage. The amount of central storage to configure to a system should be chosen at the point where an additional increment of central storage would improve system capacity by less than several percent. Reference 3 provides a methodology for estimating central storage configurations.

Figure 5 Storage size versus I/O rate



Data placement

Data placement within the storage hierarchy is automatically managed by the operating system, the subsystems, and the hardware. The systems programmer controls the eligibility of data to be retained at a particular level. Essentially, all data are initially placed in the lowest level of the hierarchy, i.e., high-capacity DASD. Based on its eligibility status, the data may be moved into and held at the DASD control unit cache level and/or the processor storage levels. Entry into the highest level, processor high-speed buffer, is strictly a hardware function resulting from a processor's request to operate on a particular data item.

A data placement strategy is implemented through the specification of the data that are eligible to be retained in processor storage and those that may be retained in the DASD control unit cache. Data are made eligible for processor storage by placing them in virtual storage. Data are made eligible for DASD control unit cache through service-level specification and placement within the DASD subsystem. (See the section on DASD control unit cache.)

The choice of which data to virtualize is based on several factors. First, can the data be virtualized? If there is no mechanism that allows a substantial amount of the data in question to be placed in virtual storage, the choice is academic. Second, do the data have an attractive access density, that is, can the movement rate to a lower level of the hierarchy be significantly reduced by retaining a manageable amount of the data? Third, are the data shared with another system? If so, management of the data to ensure that each system has an up-to-date copy may prohibit significant I/O reduction. The types of data that can be attractive candidates for processor storage include: program libraries, other libraries with frequently read members, nonsequentially accessed databases (such as those used by IMS, CICS, and DB2), and temporary files. By placing data in processor storage, the read activity (READ I/O) from the lower levels of the hierarchy is generally reduced. However, in the case of temporary files both READ and WRITE accesses can be eliminated. Reference 4 provides an overview of the data eligible for processor storage.

There are effectively no restrictions on the type of data that may reside in the DASD control unit cache. In addition, several features of the DASD control unit cache (DASD FAST WRITE, DUAL COPY) address areas beyond read-access performance for permanent data. Thus, the choice of which data to make eligible to this level of the hierarchy is based on the following system characteristics: the performance requirements for READs and WRITEs, the access density to the data; the availability requirements of the data; and the cross-system sharing characteristics of the data. The types of data that are attractive candidates for DASD control unit cache include the following: frequently read data that was not appropriate to be held in processor storage; data with a need for frequent and/or fast updates (WRITE I/O); data with high availability requirements, and data being updated by more than one system.

Concluding remarks

A storage hierarchy is the natural system structure to take best advantage of the total set of available storage technologies. It provides a sophisticated and elegant solution to the key problems of storage performance and cost. Storage hierarchies have been successfully utilized in addressing the problems of a

speed mismatch between the CPU and DASD and between the CPU and real storage. The value of the hierarchy has been recognized in large-scale processor systems for quite a while and is now being utilized to some extent in both medium and small-scale processing systems. Virtual storage is the key to making the physical storage subsystem transparent to the users of the system. New facilities available with MVS/ESA such as data spaces and hiperspaces provide the structures that allow easy and almost unlimited growth of virtual storage capacity. This in turn provides the maximum value from the storage hierarchy. The future will include ultra-large processing systems with an ever-increasing requirement for on-line data capacity that can be accessed both quickly and efficiently. In this environment, storage hierarchies will become even more important. Because they will result in higher performance, they will be used more extensively.

Cited references

1. R. E. Matick, "Impact of memory systems on computer architecture and system organization," *IBM Systems Journal* **25,** Nos. 3/4, 274–305 (1986).
2. C. P. Grossman, "Cache-DASD storage design for improving system performance," *IBM Systems Journal* **24,** Nos. 3/4, 316–334 (1985).
3. G. M. King, "Processor Storage Estimation," *Proceedings of CMG '88*, International Conference on Management and Performance Evaluation of Computer Systems, Dallas, TX, December 12–16, 1988; Computer Measurement Group, pp. 1044–1057.
4. G. M. King, "Processor Storage Overview," *Proceedings of CMG '88*, International Conference on Management and Performance Evaluation of Computer Systems, Dallas, TX, December 12–16, 1988; Computer Measurement Group, pp. 1036–1043.

Edward I. Cohen *IBM Data Systems Division, P.O. Box 950, Poughkeepsie, New York 12602.* Dr. Cohen is a senior programmer and manager of the System Performance Analysis department. He joined IBM in the Poughkeepsie Development Laboratory in 1978. He has held various positions in the area of large-system design and performance analysis. He has received two Outstanding Technical Achievement Awards, one for MVS performance analysis in 1982 and another for the design of the MVS/XA True Ready Queue Dispatcher in 1986. Dr. Cohen received a B.S. in physics from Rensselaer Polytechnic Institute in 1972 and a Ph.D. in computer science from the Ohio State University in 1978.

Gary M. King *IBM Data Systems Division, P.O. Box 950, Poughkeepsie, New York 12602.* Mr. King is a senior programmer in the System Performance Analysis department. He joined IBM in 1974 as an associate programmer with the Systems Development Division. Mr. King has been involved in the design and evaluation of the MVS resource managers, especially in the area of storage management. He received Outstanding Technical Achievement Awards for the development of a technique to study virtual storage page reference patterns and for the design of expanded storage management in 1982 and 1985, respectively. Mr. King received a B.S. in mathematics from the State University of New York at Albany in 1972 and an M.S. in computer science from the Pennsylvania State University in 1974.

James T. Brady *IBM General Products Division, 5600 Cottle Road, San Jose, California 95193.* Mr. Brady joined IBM in 1961 as a systems engineer—scientific. He has held various positions in the Data Processing, Advanced Systems Development, Systems Products, Data Systems, and General Products Divisions. He was manager of DSD systems technology and strategy, where he initiated the development of the Enterprise Systems Architecture and expanded storage. Mr. Brady is currently product manager of GPD storage systems strategy and architecture, where he is responsible for performance analysis, architecture, and the development of new product opportunities. Mr. Brady graduated from Creighton University in 1961, with a B.S. degree in mathematics. He has done graduate work in mathematics and business administration and is a graduate of the IBM Systems Research Institute.

Reprint Order No. G321-5348.