DIRECTIONS FOR MEMORY HIERARCHIES AND THEIR COMPONENTS: RESEARCH AND DEVELOPMENT*

Alan Jay Smith**

University of California, Berkeley

ABSTRACT

The memory hierarchy is usually the largest identifiable part of a computer system and making effective use of it is critical to the operation and use of the system. We consider the levels of such a memory hierarchy and describe the state of the art and likely directions for both research and development. Algorithmic and logical features of the hierarchy not directly associated with specific components are also discussed. Among the problems we believe to be the most significant are the following: (a) evaluate the effectiveness of gap filler technology as a level of storage between main memory and disk, and if it proves to be effective, determine how/where it should be used, (b) develop algorithms for the use of mass storage in a large computer system, and (c) determine how cache memories should be implemented in very large, fast multiprocessor systems.

*Partial support for this research has been provided by the National Science Foundation under grant MCS75-06768, and the Department of Energy under contracts W-7405-ENG-48 (to LBL) and EY-76-C-03-0515 (to SLAC).

**Computer Science Division, EECS Department, University of California, Berkeley, California 94720. The author is also on the staff of the Lawrence Berkeley Laboratory and a visitor at the Stanford Linear Accelerator Center.

I. INTRODUCTION

Contemporary large computer systems frequently employ a memory hierarchy such as that in figure 1, where we show cache memory, main memory, drums, disks and tape. The efficient use of this memory system is crucial to the operation of the whole computer system. In this paper, we shall examine the memory hierarchy both overall and with respect to its components in an attempt to identify research problems and project future directions for development.

The effects of the design of the memory hierarchy can be considered to fall into two (overlapping) areas: performance and logical view. Performance denotes those aspects of the hierarchy design which affect the measures of performance of the computer system, such as throughput, speed, response time, turn around time and cost effectiveness. Logical view refers to the logical view given the user of the memory system: how is the memory addressed?, named?, where is the information? (virtual vs. real location), how is this information protected?, etc. These two aspects interact, since performance is impacted by the logical view, and the cleanliness or uniformity of the logical view is often impaired by attempts to easily allow the user to tune the system to improve its performance.

By far the most fertile direction for new results (research or development) is in the study and design of memory hierarchies of the future, rather than in the optimization of current systems.

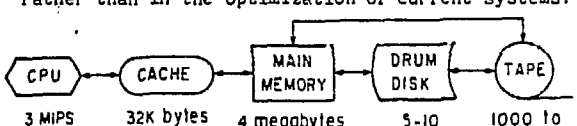

Figure 1

In figure 2, we show what we believe represents the type of large computer system memory hierarchy that will become common in the early 1980's. To figure 1, one will note that we have added in figure 2 two levels: gap filler technology and mass store. Currently, there are orders of magnitude difference (the access gap) in both cost and performance between random access (MOS) memory and mechanical storage devices such as drums or disks. Much time and effort is expended in most computer systems in finding efficient ways to accomplish the necessary transfers of information accross the access gap. A computer system using a level of storage whose technology occupies the access gap could benefit significantly in both improved performance and decreased system complexity.

Three technologies now under development and/or production fall into the middle of the access gap. Charge coupled devices (CCD's) [2,46] are fast semiconductor shift registers which shift packets of charge; they should be from 3 to 5 times cheaper [18] than main memory. Magnetic bubbles are similar shift registers [5,21] but function by shifting magnetic domains. Magnetic bubbles may be less expensive than CCD's, depending on the mechanism finally chosen (e.g. bubble lattice files [5], will be very cheap), but they are considerably slower. The third possibility is electron beam accessed memory (EBAM) [19]. Small charge patterns are stored on the face of a CRT, using a semiconductor target and hyperfine beam focusing. Only one EBAM development effort exists at the moment [43] and there is some doubt about its eventual success, since the engineering problems appear to be formidable. All three of these technologies have the potential for inclusion in memory hierarchies over the next few years.

The second difference between figures 1 and 2 is in what we call "mass store", which is a non-manual, integrated level of storage at the bottom of the hierarchy, with the potential for upwards of a trillion bits of information. Devices in this class include IBM's 3850 tape library [16], Precision Instrument's Unicon [3], IBM's photostore [23], CDC's 38500 tape library and Ampex's Terabit Memory [47]. The difference between tape and mass store is that between manual and automatic; mass storage is an integrated part of the memory hierarchy.

Both gap filler technology and mass storage devices are only just becoming generally available, and in both cases, there are numerous problems to be identified, studied and solved, in contrast with more mature technologies and devices.

In the remainder of this paper, we shall consider each of the aspects of the memory hierarchy and point out what we consider to be the open problems and likely solutions, both with regard to research and development. Sections will be devoted to each of the levels of storage

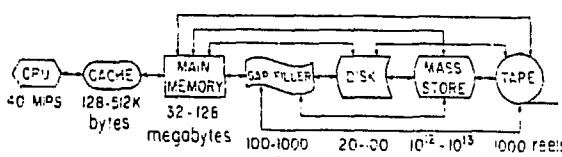

Figure 2

indicated in figure 2, as well as to those problems that don't seem to be associated with specific levels in the hierarchy.

II. CACHE MEMORIES

Cache memories, also known as High Speed Buffer memories, are very fast buffer memories managed by the hardware and placed between the CPU and the main memory. Because of the principle of locality [9], which states that (a) information in current use is likely to be used again soon and (b) information near the current locus of reference is likely to be referenced soon, cache memories are very effective. A program running in user state on a large machine will typically find about 97% to 99% of its memory references satisfied by information contained in the cache memory [27,28]. Even though the cache memory in high end machines is very fast, the processor logic is even faster; thus, large high speed computer systems are memory speed limited. Because of this requirement for high speed, the implementation details of the cache are at least as important as the more general algorithmic features of the design. Our discussion will trend toward the latter, but the importance of the former should not be neglected.

Cache memories will become larger and faster, and will appear on more and more machines. The speed of the cache is dependent on circuit technology, which is improving, and on physical size, which lower bounds propagation time. The capacity of cache memories is also limited by two factors: cost and physical size (cabinet and board space). Projected increases in density and circuit speed should aid in solving all of these problems. The largest cache memory to be found in an IBM compatable machine is the 64K byte cache in the 3033 processor, first delivered this year. If recent trends continue, this maximum capacity can be expected to double about every 3 years.

Simple cache memories are now appearing in small minicomputers. Some micros already have some of their addressable memory located on the same chip, which makes it more quickly accessable and much like a cache memory, although it is not architecturally transparent. It seems clear that as soon as circuit technology permits (1980?), small hardware managed cache memories will appear on high end microprocessors (off chip access is slower, even for the same technology, than on chip access). This represents a dramatic change from as recently as 10 years ago, when the introduction of a cache memory on the IBM 360/85 [25] was a major advance in computer architecture.

The performance issues in cache memories concern two goals: maximizing the probability of finding needed information in the cache and minimizing the time to access it if it is there. Most of the published research concerns the former. The work on cache mapping algorithms [6,35] is concerned with the first problem. A subset of the cache memory is always searched (in some sense associatively), and the problem is to select the extent of the search. If the address can map to a large number of locations in the cache, there is a higher probability of finding it, but looking takes longer. This is a well understood problem (see [22] for some data) and set sizes of 2 to 8 are commonly chosen. Selecting the size of the information transfer unit (line size) is also a well understood problem; line sizes of the order of 32 bytes (8-64 bytes) seem to be standard. Prefetching information before it is needed [36] is quite useful for cache memories, although it is not generally implemented.

The access time issue, mentioned above, leads to two possible changes in cache architecture, neither of which has been fully evaluated in the published literature. The cache is generally used for both instructions and data. Instructions are accessed by the instruction fetch and decode (I) unit of the CPU, whereas the data is used by the execution (E) unit. The I and E units are relatively separate, can both be simultaneously active, and are usually physically removed from each other in the CPU. If each of the I and E units had their own cache, access time could be decreased and bandwidth to the cache increased. The problem is that the same piece of information may be in both the instruction and data caches (especially in current architectures, where instructions can be modified), and thus consistency is a problem. This consistency problem is the same one that occurs for multiple CPU's, each with its own cache, and can be dealt with in the same way as discussed below. Software strategies can also be used. Two computers, the S-1 [26] and IBM's 801 [13], both of which represent brand new architectures, have implemented a split cache. The effectiveness of this idea has been studied only once in the published literature [40]; both that work and work by the author show that there is a very significant penalty in such an organization in terms of increased miss ratio. Further work is required, though, to see if such an organization is desirable because terms of its access time advantages. In particular, the miss ratio increase (vs. total bytes of available cache) needs to be quantified, the consistency problem needs to be looked at and the relative size of the two (I/E) caches needs to be determined.

Most large computer systems have virtual memory, by which the (virtual) addresses used by the process are mapped into real physical main memory locations. This is done conceptually by page and segment tables, but to speed access, a Translation Lookaside Buffer (TLB) is employed. The TLB maintains the correspondence between recently used virtual and real memory addresses, so that the segment and page tables seldom have to be referenced. The cache memory in current machines is accessed using a real address, which implies that every cache memory access requires prior virtual to real translation through the TLB. To a large extent the translation and lookup can occur in parallel (all of the relevant lines of the cache are read out initially using the virtual address, and then a selection is made among them using the now available real address [1]) but some time is still wasted. A possibility that has not yet been carefully evaluated in the published literature is that of a virtual address cache, in which virtual addresses are used to access the cache and real addresses are used only to access main memory. There are problems with this approach (e.g. multiple virtual addresses which map to the same real address), so further research is called for.

When a write operation occurs, information is changed and this change must eventually be reflected in main memory. This can be accomplished by (a) writing to cache memory and later copying back to main memory, (b) writing simultaneously to both cache and main memory or (c) writing to main memory only and destroying whatever copy may exist in the cache. The first strategy seems to be the most efficient [32,37], but it results in two or more different copies of the same information. This wouldn't be a problem if all references to that information used the same cache, but that is not necessarily the case in systems with channels or multiple processors. An important problem, yet to be definitively dealt with, is to design a scheme for maintaining memory consistency in a multiprocessor system where each processor has its own cache. IBM solves this problem by sending all stores to both CPU caches [20], which is only feasible for two processor systems; the bandwidth of each cache is insufficient for a larger system. The S-1 computer [26] avoids this problem by enforcing correct operation through the software. Tang [45] has proposed a scheme whereby the main memory forces consistency by keeping track of which caches contain what information. Other methods are also possible, such as one in which each cache maintains a record of what information is potentially shared. There is no generally accepted solution to this problem and further work needs to be done.

A major problem with cache memories is address space swapping during task switching. When the processor switches processes, the locality of reference generally changes abruptly and completely, with the result that the information currently in the cache is no longer in use and memory locations accessed by the new process will not be cache resident. This is the problem considered by Easton and Bennett [12] who discuss the difference between warm start (full cache) and cold start (empty cache) miss ratios. Very little has been done to see what the effect of this problem is in real systems or to study how to minimize it. One possibility is two caches - one for user state and one for supervisor state.

In cache memories, therefore, most of the "hit

ratio problems" - set size, line size, prefetching and cache size, are generally well understood. The remaining problems lie in the access time and organization area: split (data/instruction, user/supervisor) cache, virtual vs. real address access and multiple cache consistency.

III. MAIN MEMORY

In early computer systems, and in fact until quite recently, main memory was a scarce and expensive resource. Thus, a very large amount of time and energy has been spent optimizing the use of main memory, with respect to segmentation, paging [4C], compaction, restructuring, etc. Within the last couple of years, main memory has become relatively inexpensive (although not free(!)). The implication of this is that while intellectually interesting, the traditional problems of creating and evaluating paging algorithms or controlling the degree of multiprogramming are no longer major factors in the context of computer system operation.

The genuine current research problems in the area of main memory have to do with architectural extensions of main memory to support the software. For example, main memory can be made "intelligent" in several ways. Locations in main memory can be tagged [15] in which case special tag bits are associated with memory locations to indicate the nature of their contents (e.g. real, integer, capability, etc.). Alternately, memory can be active, as in Dennis' data flow architecture [11], in which case memory consists of both processing elements and storage. A third possibility is to make memory associative, as has been suggested for some data base machines (e.g. [30]). Overall, the difficulty with all of these schemes is that while main memory is becoming relatively inexpensive, none of these alternatives would be at all cheap; therefore there is the need to show significant benefits from some sort of intelligent memory before a commercially viable design will appear.

There are other minor changes that will occur in main memory design. It is likely that some hardware will eventually be added to aid in paging, as has been suggested by Denning [10] and Morris [29]. Main memory may become more complex in order to solve the cache consistency problem that was discussed in the last section. These last points are minor items, though, and represent implementation decisions, not research problems.

IV. GAP FILLER TECHNOLOGY USE

As we noted earlier, two gap filler technologies have recently been introduced. Because of the newness of these technologies, this is an extremely fertile area for study, and there is very little useable information published regarding the topics we discuss below.

Will gap filler technology will have any real place in large computer systems [24,31]? The access time gap has existed since the earliest computers, and multiprogramming currently serves quite adequately to keep the CPU busy while transfers of information across the access gap occur. Research in progress by the author indicates that a gap filler device will be very useful in computer systems by the early - mid 1980's and will become necessary by the late 1980's. If one assumes that disk access times remain fixed (see below), it can be shown (using queueing network models or simulations) that as the CPU becomes faster, the degree of multiprogramming has to be increased to maintain full CPU utilization. This process of increasing the degree of multiprogramming runs into two limits as the CPU becomes very fast: (a) the size of the disk system may not be sufficient to permit enough I/O operations to occur in parallel (this problem can be lessened in the case of sequential files by increasing the block size) and (b) a high degree of multiprogramming implies a great deal of main memory, which although it may not be very expensive, is still not free. Further, the number of disk spindles may decrease with increasing disk densities [17]. The result is that while gap filler technology isn't necessary in current computer systems, it will serve to improve cost/performance in very fast future systems for two reasons: (a) it will allow the degree of multiprogramming to be decreased, with a consequent saving in main memory cost and (b) it will relieve the bottlenecks in the disk system that will occur when the degree of multiprogramming approaches or exceeds the number of disk spindles.

The next question that arises is "which gap filler technology?" - CCD's, bubbles or EBAM? CCD's are available, are very fast, will be limited to being 3 - 4 times less expensive per bit than main memory and are volatile. Bubbles are still expensive, are available in small quantities only, are relatively slow, but are not affected by power failures. Further, future bubble technology developments (such as the bubble lattice file [5]), may make bubbles a lot less expensive than CCD's. EBAM is a technology that may or may not ever reach the market, and isn't currently available. It isn't clear which technology will eventually be dominant; no definitive answer will be available (if one exists) until the technologies in question have matured to a greater extent than currently.

If gap filler technology (GFT) is to be used in a computer system, the questions of "how" and "where" arise. There seems to be four types of "how": (1) A separate, stand alone device (e.g. a drum replacement) could be built and used as would any other device. (2) A GFT device (GFTD) could be used, either with hardware or software management, as a dynamic file migration device - as files were opened, they would be moved to this device. (3) GFT could be used as an extension of main memory. The memory address generated would refer to the GFT and what we call main memory currently would be the lower (slower) of two levels of cache. (4) GFT could be used as a cache for I/O streams. Tracks or cylinders of a disk could be buffered dynamically. Measurements available to the author (to be published eventually) show that alternative #1 above performs relatively poorly and alternative #4 very well. Measurements will soon be available for #2, thus completing the currently possible comparisons. There doesn't seem to be any way to directly evaluate the performance of #3, since there isn't any obvious way to determine how a system would behave if addressable main memory (i.e. the address space) increased by an order of magnitude or more, while "fast" memory didn't.

The use of GFT as a dynamic I/O buffer (#4) raises the question of "where"? - should the buffering be placed in the disk spindle, at the disk controller, attached to each channel or be global to the whole I/O system? It is obviously more efficient in terms of hit ratio to place the buffering (cache) as close to the CPU as possible, but this raises questions of architectural transparency - the closer the buffering is to the CPU, the more likely it is to impact the CPU architecture or the operating system. Buffering placed in a disk spindle [39] or controller could be invisible to the CPU and such buffering could be manaufactured by an independent manufacturer. Memorex [6] has in fact just announced such a buffered disk.

However and wherever the GFT is used, there are some associated research problems: (1) Should the storage be managed by the software or hardware in terms of search, replacement, placement, etc? (2) What are the migration algorithms to be used to move information to/from the GFT? (3) What bandwidth is required? (4) What access time is acceptable? (5) What should the transfer unit or blocksize be? (6) How should the storage be organized? (7) How are multiple copy problems (as noted above in the discussion of cache memories) resolved? (8) What are the implications for disk architecture, channels, main memory interface, etc.

We should point out here the important difference between technical and commerical considerations. It is relatively easy in some cases to show that certain implementation strategies are more convenient for the user and/or yield better performance, but there are the problems of market and compatability. An existing systems manufacturer usually has to introduce a device which is compatable with current product offerings and which requires minimal changes in hardware and software. An independent manufacturer generally has to introduce a device which can just be plugged in as a replacement for something that already exists. The only case in which decisions can be made solely on the basis of being "right" is in the case when the system is completely new. Thus among the likely product offerings of the next five years there should be: a stand alone gap

filler device and a buffered disk (e.g. [8]), buffered either in the spindle or controller. Less likely (and later to appear) are GFT main memory [illegible] or global system I/O buffers, for the reasons noted above.

V. DISKS AND DRUMS

Disks, like main memory, are a mature technology, and there are few new problems or developments associated with them. A good survey of disk technology appears in [17]. Disks have been increasing in density fairly steadily; currently the density seems to be doubling about every 3 to 5 years. None of the manufacturers (to the author's knowledge) are projecting any but the most minor performance (access time) improvements. One could expect to see perhaps 10% to 30% decreases in access time over the next 3 to 10 years, which is a small enough factor that it will have little impact on system performance or operation.

Drums are no longer cost effective with the decreases in cost for main memory and disk and the appearance of gap filler technology.

Some of the research activity in disks/drums has been in analytic modeling, and [38] provides a (1976) bibliography of the disk literature. There is modeling work still to be done in evaluating the effectiveness of the inclusion of gap filler technology in the disk system [39] as was discussed in the last section.

There are a number of techniques which have been implemented in one or more systems, but which merit further development, publicity, and/or research. The disks used for the Cray I at Los Alamos [33] have a buffer in the controller so that transfers can be buffered ahead or behind. In this same system the I/O system has been optimized with "strategy routines" in two aspects: (a) file placement in consecutive sectors is encouraged while minimizing fragmentation and (b) file transfer delays are minimized by reading ahead and writing behind. Further research is needed in the design of such strategy routine algorithms.

One possible direction for disk development is to make disks "intelligent", since the cost of logic is declining rapidly with respect to the cost of the mechanical components of the disk. This is being done with drums, as noted earlier, in the RAP (Relational Associative Processor) system at the University of Toronto [30]. In that case, the storage device is programmed to search for the desired record. Similarly, the disk could be allowed to do its own error correction [14], a task which is currently allocated to one or more of the controller, channel, or CPU.

Overall, therefore, we see only three real research problems having to do with disks: intelligent disks, disks associated with gap filler technology, and disk (or I/O system) strategy routines. The remaining issues are ones of either straightforward development or more extensive implementation and publicity. The directions for development are much the same as those for research: more logic (intelligent or not) will appear in the disk spindle or controller and this logic will serve to operate, correct errors and buffer the disk.

VI. MASS STORAGE

As noted earlier, mass storage devices (storage on the order of a trillion bits or more) have finally achieved commercial acceptance. Associated with this new capability are of course some new and interesting research problems.

All current and projected main storage devices have long access delays; thus it is important to keep frequently used data sets resident on faster devices. We call the problem of deciding when to move information from mass store to disk and later from disk back to mass store, the file migration problem. The only published useful study to date is by Stritter [44] (also see [34]); the author has submitted research on this subject for publication [41,42]. Both Stritter and the author are concerned with data and file access patterns over many months. Files would migrate after a period of inactivity of several days or weeks. There is also a need for research on file migration over short (minutes to hours; main memory <--> GFT <--> disk) and intermediate term (hours to days; GFT <--> disk <--> mass store). Research on these topics is in its very early stages by the author and his students; no other work seems to be in progress or available.

Mass storage technologies are not usually random access, and often are not even erasable. The photostore [23] uses nonerasable photographic chips; the Unicon [3] burns holes in metalized strips. Even the tape based devices, while erasable, do not usually accept in-place updates. One therefore has a whole new set of problems in the management of this level of storage: how should unused or no longer valid space in the storage media be reclaimed?, how/when can the data be compacted?, should the information be moved around so that data used together is physically stored together? Management policies such as these have to be developed for each of the types of technologies in use for mass storage.

The vast capacity of mass storage creates new and severe problems in terms of reliability and back-up. The mass storage device is far too large to dump periodically. It may or may not be feasible to make incremental dumps, duplicate important data sets and make special copies of vital data sets on tapes which can then be removed for safekeeping. Extra care must be taken to be sure that the loss of directory or map information does not result in the loss of much or any data - directories can be duplicated and files/blocks/records can be self identifying. Carefully thought out schemes for the integrity of mass storage information must be developed. The use of a separate file system controller (as is used in the IBM 3850) may facilitate solutions to these problems.

Over the next few years, the most visible changes that will occur in the mass store area will be in the development of larger and faster mass store devices. Less visible but no less important will be improved operation for these devices in terms of algorithms for migration, file placement, reliability and recovery. Also, we can expect to see somewhat more intelligent system software, which isolates the user from the details of the device to a greater extent than at present.

VII. LOGICAL and USER VIEW PROBLEMS

One of the most important problems in a memory hierarchy, and one which is not associated with any one type of technology or level is the user's view of the memory hierarchy. It was proposed many years ago (e.g. [7]) that the user be given a very large virtual address space, sufficient to encompass not only main memory, but the entire program and data space of his process. This virtual address would be mapped dynamically by the system to the physical storage, and the user would be encouraged to remain unaware of the physical location and attributes of the data. There is nothing new in this idea, yet despite its obvious (to the author) advantages, it has been implemented to only a very limited extent in most systems. An important and pressing development problem is the introduction of such logical/physical independence for the memory hierarchy into new or existing operating systems.

Despite the comments in the above paragraph, there is some question as to how extensive the address space should actually be. Is it reasonable, for example, to make mass storage byte addressable, with the consequent cost of 40 or more bits per address? There is clearly the need for a means of dealing with mountable volumes (tapes, disk). It is probably desirable to allow for dynamic mapping from one set of logical names (directory and file names) to another (binary virtual addresses). Synonyms (many virtual address will map into the same physical location) will probably occur, and there are problems involved in determining how/when synonyms should exist and how they should be handled because of problems with consistency. Structures for name spaces, such as directory structures, have been studied but without definite conclusions; there is room for further work.

Another aspect of memory hierarchies that has not been fully developed is the interface between operating systems and data base systems. A data base system can be considered to be a powerful command (query) language on top of a very sophisticated file system. Both command languages and file systems are part of operating systems;

there is *considerable need* to integrate the two. The management of the physical storage of the data base system should perhaps be subsumed into the overall management of the unified memory hierarchy on which the data base system runs.

VIII. CONCLUSIONS

In this paper we have explored the structure of the computer system memory hierarchy with an eye to noting aspects of such hierarchies which are not fully understood, developed or optimized. We found that there is important research still to be done on cache organization, but not on the traditional miss ratio problems of cache design. With regard to gap filler technology, almost every aspect of its use poses both research and implementation problems - should it be used:, where?, when?, how? Similarly for mass storage - how should it be operated? how/when should files be migrated? what should be done about reliability and recovery? Those parts of the memory hierarchy of long standing, such as main memory, disks and drums pose few new problems. The principal research problem for those levels is: how (or if) should these parts of the system be made intelligent? With the decreasing cost of logic, intelligent memory or peripherals becomes a viable alternative. We also noted that there are questions about the logical structure of the user name space, and how it should map into the memory hierarchy. All of these problems are both interesting and important, and we hope that this paper will stimulate some research in these directions.

BIBLIOGRAPHY

[1] Amdahl Corp., "470V/6 Machine Reference Manual", Amdahl Corp., Sunnyvale, Ca. 1976.
[2] G. Amelio, "Charge-Coupled Devices for Memory Applications", Proc. NCC 1975, pp. 515-522.
[3] C. H. Becker, "Unicon Computer Mass Memory System", Proc. FJCC 1966, pp. 711-716.
[4] James Bell, David Casasent and C. Gordon Bell, "An Investigation of Alternative Cache Organizations", IEEE Trans. Comp., C-23, 4, April, 1974, pp. 346-351.
[5] A. Bobeck, F. Bonyhard and J. Geusic, "Magnetic Bubbles - An Emerging New Memory Technology", Proc. IEEE, 63, 8, pp. 1176-1195, August, 1975.
[6] C. J. Conti, "Concepts for Buffer Storage", IEEE Computer Group News, March, 1969, pp. 9-13.
[7] R. C. Daley, and P. G. Neumann, "A General Purpose File System for Secondary Storage", Proc. FJCC, 1965, pp. 213-229.
[8] Datamation, June, 1978, p. 254.
[9] Peter J. Denning, "On Modeling Program Behavior", Proc. SJCC, 1972, pp 937-944.
[10] Peter Denning, "The Working Set Model for Program Behavior", CACM, 11, 5, May, 1968, pp. 323-333.
[11] J. B. Dennis, "Proposed Research on Architectural Principles for Large Memory Systems", Computation Structures Group Memo 132, Project MAC, Massachusetts Institute of Technology, Cambridge, Mass. October, 1975.
[12] M. C. Easton, and B. T. Bennett, "Transient Free Working Set Statistics", CACM, 20, 2, pp. 93-99, Feb. 1977.
[13] Electronics, "Altering Computer Architecture is Way to Raise Throughput, Suggests IBM Researchers", Electronics, December 23, 1976, pp 30-31.
[14] Mark Feller, "Intelligent Disk: The Next Generation", Proc. IEEE Computer Society Conference, September, 1977, pp. 306-310.
[15] Edward Feustel, "On the Advantages of Tagged Architecture", IEEETC, C-22, 7, July, 1973, pp. 644-656.
[16] John P. Harris, R. S. Rohde and N.K. Arter, "The IBM 3850 Mass Storage System: Design Aspects", Proc. IEEE, 63, 8, August, 1975, pp. 1171-1176.
[17] Kenneth Haughton, "An Overview of Disk Storage Systems", Proc. IEEE, 63, 8, August, 1975, pp. 1148-1152.
[18] David A. Hodges, "A Review and Projection of Semiconductor Components for Digital Storage", Proc. IEEE, 63, 8, August, 1975, pp. 1136-1147.
[19] W. C. Hughes, C. Lemmon, H. Parks, G. Ellis, G. Possin, and R. Wilson, "A Semiconductor Nonvolatile Electron Beam Accessed Mass Memory", Proc. IEEE, 63, 8, pp. 1230-1240, August, 1975.
[20] IBM, "System/370 Model 168 Theory of Operation/Diagrams Manual (Volume 4) - Processor Storage Control Function", IBM Maintainance Library SY22-6934-3, May, 1975.
[21] J. Egil Juliussen, David M. Lee and Gerald Cox, "Bubbles Appearing First as Microprocessor Mass Storage", Electronics, Aug. 4, 1977, pp 81-86.
[22] K. R. Kaplan and R. L. Winder, "Cache Based Computer Systems", Computer, March, 1973, pp. 30-36.
[23] J. D. Kuehler and H. R. Kerby, "A Photo-Digital Mass Storage System", Proc. FJCC, 1966, pp. 735-742.
[24] Glenn Langdon, "A Figure-of-Merit Approach to a Two Level Hierarchy for a Disk", IBM Research Report RJ 1778, May, 1976.
[25] J. S. Liptay, "Structural Aspects of the System/360 Model 85, II The Cache", IBM Systems Journal, 7, 1, 1968, pp. 15-21.
[26] Thomas McWilliams, Lawrence Widdoes Jr. and Lowell L. Wood, "Advanced Digital Processor Technology Base Development for Navy Applications: The S-1 Project", Lawrence Livermore Laboratory report UCID-17705, September, 1977.
[27] Barry Merrill, "370/168 Cache Memory Performance", Computer Measurement and Evaluation Newsletter, SHARE, July, 1974, pp. 96-103.
[28] G. Milandre and R. Mikker, "VS2-R2 Experience at the University of Toronto Computer Center", Session Report, 44'th SHARE Meeting, Los Angeles, Ca., pp. 1887-1895.
[29] James Morris, "Demand Paging Through Utilization of Working Sets on the MANIAC II", CACM, 15, 10, October, 1972, pp. 867-872.
[30] E. A. Ozkarahan and K. C. Sevcik, "Analysis of Architectural Features for Enhancing the Performance of a Data Base Machine", ACM Transactions on Data Base Systems, 2, 4, December, 1977, pp. 297-316.
[31] A. Pohm, "Cost/Performance Perspectives of Paging with Electronic and Electromechanical Backing Storages", Proc. IEEE, 63, 8, pp. 1123-1128, August, 1975.
[32] Arthur V. Pohm, Om. P. Agrawal and Ronald N. Monroe, "The Cost and Performance Tradeoffs of Buffered Memories", Proc. IEEE, 63, 8, August, 1975, pp. 1129-1135.
[33] Michael Powell, "The DEMOS File System", Proc. Sixth ACM Symp. on Operating Sys. Principles, Nov. 1977, pp. 33-42.
[34] Ron Revelle, "An Empirical Study of File Reference Patterns", IBM Research Report RJ 1557, April, 1975.
[35] Alan Jay Smith, "A Comparative Study of Set Associative Memory Mapping Algorithms and Their Use for Cache and Main Memory", IEEETSE, SE-4, 2, March, 1978, pp. 121-130.
[36] Alan Jay Smith, "Sequential Program Prefetching in Memory Hierarchies", April, 1977, to appear, Computer (1978?).
[37] Alan Jay Smith, "Characterizing the Storage Process and Its Effect on the Update of Main Memory by Write-Through", January, 1976, to appear, JACM (1978?).
[38] Alan Jay Smith, "Analysis of a Locality Model for Disk Reference Patterns", Proc. Second Conference on Information Sciences and Systems", The Johns Hopkins University, Baltimore, Md., April, 1976, pp. 593-601.
[39] Alan Jay Smith, "On the Effectiveness of Buffered and Multiple Arm Disks", Proc. Fifth Computer Arch. Symp., April, 1978, Palo Alto, Ca., pp. 242-248.
[40] Alan Jay Smith, "Bibliography on Paging and Related Topics", August, 1978, submitted for publication.
[41] Alan Jay Smith, "Long Term File Migration, Part I - File Reference Patterns", August, 1978, submitted for publication.
[42] Alan Jay Smith, "Long Term File Migration, Part II - File Replacement Algorithms", August, 1978, submitted for publication.
[43] D. O. Smith, "Electron Beam Accessed Memory", Proc. IEEE Computer Society Conference, February, 1978, pp. 167-169.
[44] Edward Stritter, "File Migration", Stanford Linear Accelerator Report SLAC-200, January, 1977.
[45] C. K. Tang, "Cache Design in the Tightly Coupled Multiprocessor System", Proc. NCC, 1976,

pp., 749-753.
Dean Tombs, "An Update: CCD and Bubble Memories", Spectrum, April, 1978, pp 22-30.
[47] Manfred Wildmann, "Terabit Memory Systems: A Design History", Proc. IEEE, 63, 8, August, 1975, pp. 1160-1165.