# Some File Structure Considerations Pertaining to Magnetic Bubble Memory

**William E. Wright**
Department of Computer Science, Southern Illinois University at Carbondale, Carbondale, Illinois 62901, USA

**Several aspects of file structures are considered with respect to magnetic bubble memory architecture. Topics discussed include record sizes, wrap-around, parallel and serial systems, blocking, tree structured files, indexed sequential files, and hashed files. Primary attention is given to minimizing access times. Various file structures are identified for which the nature or extent of utilization is significantly different for magnetic bubble memories than for rotating disks or drums. Some structures are shown to be suitable or unsuitable for magnetic bubble memories. Tendencies of structure parameters to be bigger or smaller for magnetic bubble memories are shown.**

## INTRODUCTION

Magnetic bubble memory (MBM) is an important new and developing architecture for computer storage.[1–3] This technology shows real promise for helping fill the gap between high speed, low capacity, high cost random access MOS memory, and low speed, high capacity, low cost rotating disk memory. Figures 1 and 2 (cf. Ref. 4) roughly illustrate the place of MBMs in this gap.

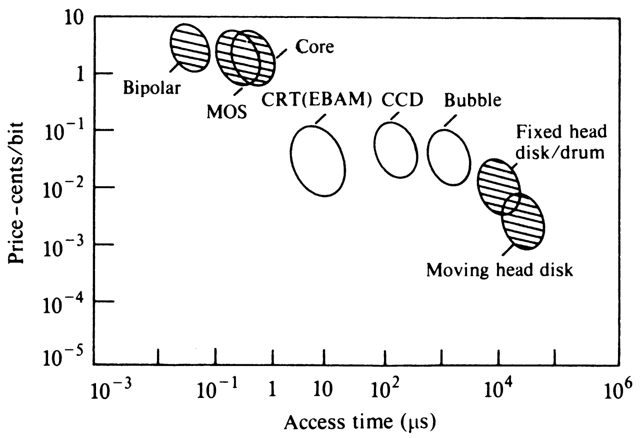

**Figure 1.** Price versus access time.

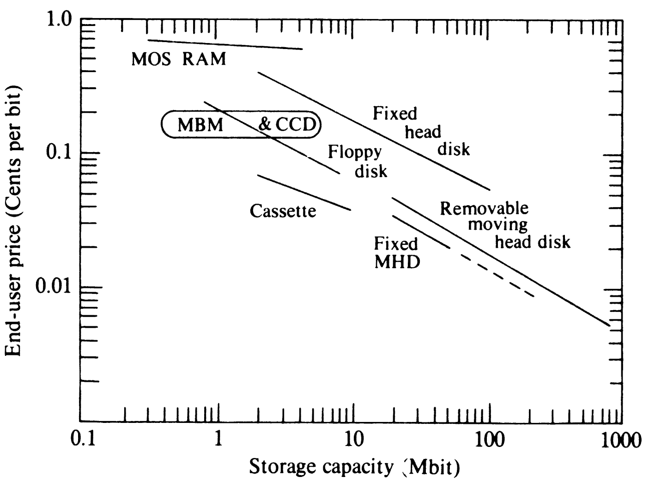

**Figure 2.** Price versus capacity.

A distinguishing feature of MBMs is the speed and manner in which they store and retrieve information. Two useful measures of this speed are the time required to reach the first bit of information (the seek time), and the time required to transfer the information from or to the device (the transfer time). Access time can be defined as the sum of seek time and transfer time. This definition of seek time is commonly referred to as seek and rotational delay in rotating disk terminology, rotational delay in drum terminology, and access time in MBM terminology.

Table 1 gives typical seek and transfer times for eight devices, MOS, RAM, a CCD chip, two MBM chips, a moving head cartridge disk, a floppy disk, a large capacity moving head disk pack, and a rotating drum. Owing to various considerations of cost, capacity, speed, volatility, etc., it appears at this time as if MOS (or core) is indisputably the best technology for high speed random access main memory, and the moving head disk pack is the best technology for large on-line secondary storage.

**Table 1. Typical access timing**

| Device | Example | Average seek time (ms) | Transfer time (μs/bit) |
|---|---|---|---|
| MOS RAM | | 0.0003 | 0.5 |
| CCD chip | TI TMS 3064 | 0.41 | 0.2 |
| MBM chip | TI TIB 0103 | 4.0 | 20.0 |
| MBM chip | TI TIB 0303 | 7.3 | 10.0 |
| Disk cartridge | DEC RL 01 | 63.0 | 0.3 |
| Floppy disk | DEC RX 01 | 360.0 | 6.0 |
| Disk pack | IBM 3350 | 33.0 | 0.1 |
| Rotating drum | IBM 2301 | 8.6 | 0.1 |

For the next five years and more, the prime application area for MBMs would seem to be for small auxiliary storage, either temporary or permanent, especially for microcomputer and minicomputer systems (cf. Ref. 1, pp. 346–347, Refs 4–7). This area is now served primarily by cartridge disks, floppy disks, and cassette tapes. Unlike CCDs, MBMs are nonvolatile and hence are suitable for permanent storage, although they would be quite expensive for off-line storage.

With these roles and performance characteristics in mind, an important problem to investigate is how to

CCC–0010–4620/83/0026–0043 $04.50

construct software for MBMs in order to best exploit the hardware. In particular, we need to consider ways of organizing information in MBMs in the most efficient manner, where efficiency is measured primarily in terms of access time and space utilization. Accordingly, the purpose of this paper is to consider several aspects of file structures as applied to MBM architectures.

## CHIP ARCHITECTURE

We shall first need to look at the organization of MBMs, using as a starting example the major–minor loop design of the TIB 0103 (a.k.a. TIB 0203) (cf. Fig. 3). During a read operation, the minor loops are shifted in unison to the proper bit position, and then they simultaneously transfer one bubble (bit) each to the major loop. The major loop then shifts the bubbles around to a replicator, with the replicates going on to a detector and the originals staying in the major loop. The replicate bubbles in the detector are decoded into 0 and 1 bits which are placed in a buffer.

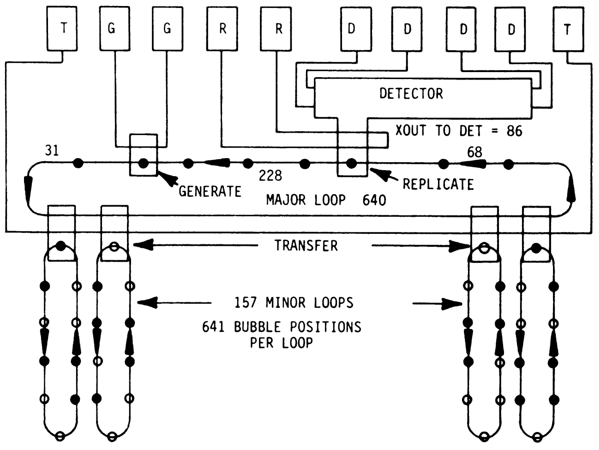

Figure 3. Diagram of TIB 0103 (cf. Ref. 1, pp. 279–281 and Ref. 15).

The major loop continues shifting the originals on around until they are simultaneously returned (transferred) to the same bit position in the minor loops which they previously vacated. The major loop (including transfer connections) is designed to have the same number of bubble positions as each minor loop, and hence the same cycle time. Thus the major loop will reach the proper position for the return transfer just when the minor loops have cycled around and returned to the vacated bubble position.

The write operation makes use of a generator component instead of a replicator and detector. The generator places bubbles in the major loop, which shifts them around to the proper position for transfer to the minor loops. The beginning of generation and the shifting of the minor loops must be synchronized so that the major loop and minor loops arrive simultaneously in the proper position for transfer.

The TIB 0303 is not exactly a major–minor loop design, using a 'block replicator' line for reading and generator and swap lines for writing (cf. Fig. 4). During a read, the block replicator generates replicates of the bubbles in the top position of the minor loops, without removing the originals from the minor loops. It then sends the replicates on to the detector, where bits are generated and the replicates are deleted (annihilated).

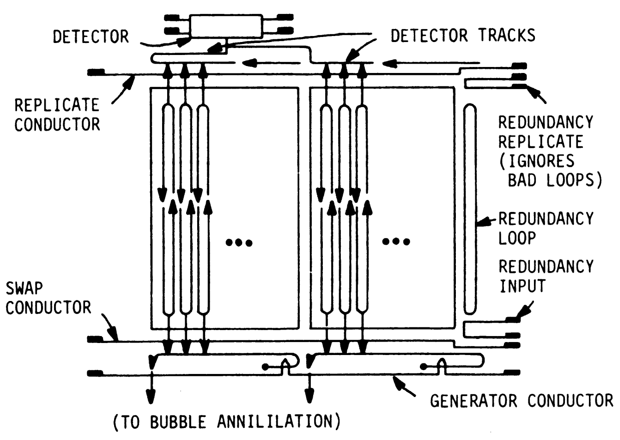

Figure 4. Diagram of TIB 0303 (cf. Refs 2 and 16).

During a write operation, a generator places bubbles into the generator line, which shifts them to the proper position for transfer into the minor loops. They are then transferred into the correct minor loop position, at the same time as the prior contents of that position are transferred (swapped) out. The swapped out bubbles are shifted along the swap line to an annihilator.

In the TIB 0103, there are 157 minor loops with 641 bubbles (bits) in each. In the TIB 0303, there are 252 minor loops containing 1137 bits each. Thirteen of the loops in the TIB 0103 and 28 of the loops in the TIB 0303 are redundant. These unused loops do not affect any of the general conclusions which follow, so we shall merely assume that they are used. Bubbles are propagated (shifted) at the rate of 10 μs/bit (bubble) in both chips. Owing to spacing constraints, the major loop in the TIB 0103 contains bubbles in only every other position, thus yielding an effective transfer rate of 20 μs/bit in the major loop.

Seek times can vary with the operation (read or write) as well as the chip. Starting with the TIB 0103, the average random minor loop shift is $(641 - 1)/2 = 320$ bits. Another 1 bit shift in the transfer components and $68 + 86 = 154$ bits in the major loop (cf. Fig. 3) yields a total average shift of 475 bits. At 10 μs/bit, the average seek time for a read is therefore 4.75 ms. The transfer time is dictated by the double spacing of the minor loops (cf. Fig. 3), which causes them to occupy $157 \times 2 - 1 = 313$ positions in the major loop. The time is therefore 313 positions × 10 μs/position = 3.13 ms. We should remark that even after seek and transfer a read operation is still not complete, since the bubbles must be returned to the minor loops. The time required for this is $(228 + 31 - 86)$ bits × 10 μs/bit = 1.73 ms. The total average time for a read operation is thus 4.75 ms + 3.13 ms + 1.73 ms = 9.61 ms, or 1.5 cycles.

A write operation requires a shift of $(2 \times 157 - 1 + 31)$ bits = 344 bits in the major loop. The minor loops must shift along with the major loop, and in fact will pass by the proper bit position if it is less than 344 bits away. There will thus be required a random minor loop shift prior to the beginning of the major loop shift, to provide synchronization. The total average time is therefore

(344 + 320) bits × 10 μs/bit = 6.64 ms. This figure includes transfer time for all bits, however, and a better measure of seek time would probably exclude a transfer time of 3.13 ms, which is the transfer time for a read. The average seek time for a write would then be (6.64 − 3.13) ms = 3.51 ms. Note that Table 1 gives a 4 ms average seek time for the TIB 0103, without distinguishing between read and write.

For the TIB 0303 we shall omit the details and assume that the 'overhead' for both reading and writing is 1.62 ms. Using an average minor loop shift of (1137 − 1)/2 = 568 bits, we then get an average seek time of 1.62 ms + 568 bits × 10 μs/bit = 7.30 ms, the figure given in Table 1.

The above discussion has served not only to explain the relevant functioning of the TIB 0103 and the TIB 0303, but also to illustrate both similarities and differences between the two designs. File structures for MBMs can be reasonably considered without looking at every possible architecture. Similarities and differences in the performance of similar architectures can generally be inferred from similarities and differences for these two examples.

There are many other possible designs for MBMs. Several major–minor loop designs are described in Refs 3 and 5, and lattice designs are described in Ref. 1, pp. 327–328 and Ref. 8. The TIB 0103 and TIB 0303 have been used as examples because they are commercially available, relatively well-known, reasonably typical, and reasonably efficient.

Besides chip architectures, there are many possible configurations for systems of chips. Except where stated otherwise, we shall assume a system of eight TIB 0103 chips connected in parallel. Thus, during a transfer operation, for example, the eight chips will simultaneously transfer the eight bits in a byte into or out of a single data buffer. Hence we can think of each position in each loop in the memory *system* as containing a single byte instead of a single bit.

## PRIMARY FILE ORGANIZATION

### Standard blocks

Now let us consider some aspects of the primary file organization for our secondary memory system. The highest accessing efficiency can be achieved using physical records of size $k$ bytes where $k$ is the number of minor loops (e.g. 157). Then one physical record can be transferred (read) by shifting the minor loops to the proper bit position, shifting one bit from each of the minor loops into the major loop, and then shifting the major loop around through one complete cycle. Writing is similarly most efficient using $k$-byte 'blocks'.

### Wrap-around

It is also very efficient to use records which are a multiple of $k$ bytes long. The TIB 0103 has a special multiple-block mode for reading and writing such records. In this mode, multiple blocks can be read or written without using a complete cycle for each, and in effect the logical record 'wraps around' from the end of one block to the beginning of the next. The blocks are spaced 322 positions apart on the minor loops, so that the transfer of a block other than the first one begins exactly 3.22 ms after the previous block began to be transferred. Allowing 3.13 ms for a block transfer therefore gives a 'wrap-around' delay of just 0.09 ms. This is obviously a major improvement over the average random seek time of 4.75 ms for read and 3.51 ms for write, as well as the sequential seek time of 1.56 ms for read and 0.32 ms for write.

Wrap-around can also be facilitated on the TIB 0303, without the use of a special multiple-block mode. This requires defining 'logically adjacent' blocks to be $d \cong 415$ positions apart. Then when the system has just finished one access (1.62 ms + 2.52 ms = 4.14 ms), the minor loops will be in almost perfect position to begin the next access. (It is necessary to use a skip of 415 instead of 414 because 414 and 1137 are not relatively prime.) Thus the wrap-around delay becomes 4.15 ms − 2.52 ms = 1.63 ms.

### Nonstandard records

It is logically possible to use record sizes different from $k$ bytes or a multiple of $k$ bytes, with some degradation in seek time. The degradation is caused by two likely results of a 'nonstandard' record size: records starting and/or stopping in the middle of the major loop, and records wrapping around from the end of one $k$-byte block to the beginning of the next. Wrap-around has already been discussed in the previous section, so we now need to consider the problem of starting or stopping in the middle of a block.

As an example, if a block to be read began in position 51 of the major loop, then there would be an additional latency during the major loop shift cycle while the first 50 bit positions were shifted and ignored. On the TIB 0103, assuming an average shift of (157 − 1)/2 = 78 positions, the average additional time would be 78 bits × 20 μs/bit = 1.56 ms. This would increase the average seek time from 4.75 ms to 6.31 ms. On the TIB 0303, the increase in average seek time would be from 7.30 ms to 8.56 ms.

The situation is compounded for writing since there is no capability for selectively writing a portion of the minor loops. The entire $k$-byte block would have to be read, the proper segment changed, and the entire block rewritten.

A destructive read could be used on the TIB 0103, followed by a properly synchronized write. The timing would in fact be the same as for a complete nondestructive read operation, with an initial random minor loop shift followed by a complete cycle shift of the major and minor loops. The average time would therefore be 3.20 ms + 6.41 ms = 9.61 ms, or 1.5 cycles.

The situation is slightly better (in cycles) for the TIB 0303, since it reads at one end of the minor loops and writes at the opposite end. After an initial random shift averaging one-half cycle, reading would require 4.14 ms and writing another 4.14 ms. The writing would begin 0.5 cycles after the reading began, hence the total time requirement would be 0.5 cycles + 0.5 cycles +.4.14 ms = 15.51 ms, or 1.36 cycles. This analysis assumes the capability of partially overlapping the read and write operations.

It is obvious for both of these cases that nonstandard writing is somewhat inefficient. Nonstandard reading could be efficient for some applications, since, although the average seek time for reading is increased, the transfer time could be decreased. Nonstandard record sizes would thus appear to be desirable for applications dominated by retrievals (reading), with relatively little updating (writing).

### Multiple-chip systems

One aspect in which bubbles differ substantially from disks is in their modularity. Bubbles are extremely flexible with regard to their ability to be combined in series or parallel. For example, we have assumed in much of this paper that eight bubble chips were combined in parallel, so as to transmit an entire 8-bit byte in one bubble shift cycle. It would also be possible to combine eight chips in series, so that an access operation first selects a chip and then performs the transfer. Obviously, any number of chips could conceivably be combined, either in series or parallel.

There are advantages and disadvantages to both types of combination. The overwhelming advantage of the parallel combination is the increase in the transfer rate. For example, eight TIB 0103 chips in parallel would have a transfer rate of 50 000 bytes/s = 400 000 bits/s, as compared to 50 000 bits/s for a single chip. Access time would not necessarily be decreased by a factor of 8 because of seek time. In general, a parallel system will decrease the transfer time by a factor equal to the number of chips in parallel, and will decrease access time by a somewhat smaller factor.

The primary advantage of a serial system is the potential independence of the different chips. In a parallel system the chips are shifted simultaneously and in synchronization, so that they all address the same logical location. In a serial system it would be possible for one chip to remain at one location while another chip was being shifted to a different location. The likely motivation would be to allow the first chip to remain optimally positioned for a subsequent 'sequential' access. For example, each chip may correspond to a logical unit during an external sort. With additional logic, it would be possible to have the chips shifting, but not transferring, simultaneously and independently, like independent seeking of multiple disk drives on a single controller. With still further logic the chips could shift and transfer simultaneously and independently. This configuration would in effect include the parallel combination as a special case, and hence would be inherently more powerful (and expensive).

There are several disadvantages of the parallel and serial combinations that should be identified. The obvious disadvantage is the cost of the logic needed to implement the combination, be it parallel or serial. The extra features suggested for the serial combination would require even further logic. The advantages of each system give rise to a comparative disadvantage for the other system, i.e. slower transfer rate for the serial system and nonindependence for the parallel system.

The parallel system has two other subtler disadvantages. One is that the increased block size may be inefficient for applications with smaller logical records. For example, 64 TIB 0103 chips in parallel would yield blocks of size 1256 bytes, so that the accessing of smaller logical records might be accompanied by the inefficiencies discussed in this section. It should be noted, however, that the inefficiencies can probably be avoided or reduced through the use of blocking on sequential files, bucketing on hashed files, or high order nodes in tree-structured files.

The other disadvantage stems from the presence of redundant loops in the major–minor loop bubble architecture. These loops are logically masked out during a read or write operation, but the timing of the operation is the same as if the loops were used. In a parallel system the bad loops would generally not be the same in each chip. The problem is that during any bubble shift cycle (e.g. 10 μs), each chip may not be able to make its 1-bit 'contribution'. One solution to the problem would be to mask out an entire 'byte' whenever any of its bits was faulty. This solution would obviously result in very poor efficiency as the number of parallel chips grew, unless the chips could be selected so as to have very high agreement in the positions of the redundant loops. A second solution would be to produce chips with 0% redundant loops, but this would of course result in higher production costs and/or lower yields.

A third solution would involve the use of extra logic on each chip to 'store up' at the beginning of a read operation extra bits to be used as replacements for subsequent faulty bits. These stored bits could then be inserted at appropriate times to allow for the uninterrupted generation of complete 'bytes' after an initial delay to allow for the 'storing up'. The delay might correspond to just 13 positions (260 μs) on the TIB 0103 or 28 positions (280 μs) on the TIB 0303, even if some of these positions were themselves redundant.

As can be seen, the bubble architecture gives rise to a number of novel possibilities with regard to ways of combining chips into systems. Some choice must be made as to the extent of parallelism to be used. The proper choice is significantly dependent on the application, in obvious ways such as record size, file organization, blocking, etc. Even the nature of some applications has implications for parallelism. For example, it can be shown that an external sort procedure will benefit from having two independent systems (corresponding to input and output), with each system being entirely parallel. For example, if there are 32 chips available, they should be configured into two 16-chip parallel systems. Any increase in independence (e.g. eight 4-chip systems), with the combined transfer rate held constant, will merely incur additional expense without reducing the processing time.[9]

At any rate, the modularity of the bubble architecture can permit substantial improvements in the performance of the secondary memory system through the proper combination of the modules.

## SECONDARY ORGANIZATION

### Introduction

The specific type of organization for a file in MBM small auxiliary storage is of course very important. Four types to consider are sequential, random, indexed sequential, and hashed.[10] Some aspects of these file types require little or no special considerations for the MBM architecture, but several other aspects do merit special attention.

## Sequential files

Certainly an important structure for small auxiliary storage is the sequential file. For the most part, the structure is relatively independent of the storage medium, but one way in which it isn't concerns blocking.

Blocking involves the reading and writing of several logical records in one physical unit (block). The purpose of blocking is to reduce seek time and reduce the number of interblock gaps. Unlike magnetic tapes or disks, MBMs need no interblock gaps. Hence this motivation for blocking is eliminated. They do have seek times, however, and a reduction in seek time is beneficial.

It should be noted, however, that the benefit is not as great for MBMs as for disks. One reason is that the seek time is much smaller for MBMs than for disks, by roughly 1 to 2 orders of magnitude. Thus the amount of seek time saved by blocking is smaller in absolute terms.

A more subtle reason is that the ratio of seek time to transfer time is much smaller for MBMs. Assuming 1000 bit blocks, for example, we get ratios of 0.2:1 and 0.73:1 for the MBM chips, 210:1 for the cartridge disk, 60:1 for the floppy disk, 330:1 for the disk pack, and 86:1 for the drum. Since seek time is a drastically smaller component of total access time, the relative benefits of reducing it through blocking are much less.

An important characteristic of MBMs can make the seek time even smaller for sequential processing, hence further reducing the benefit of blocking. The characteristic is the capability of instantly stopping the movement of the bubbles. If the bubbles are stopped after completion of a read, then the next read can begin right where the minor loops stopped, without necessitating a random shifting of the minor loops.

For the TIB 0103, we thus get a read seek time of 0.01 ms (transfer) + 1.54 ms (major loop shift) = 1.55 ms. For the TIB 0303, we get simply 1.62 ms for the seek. Similar results would follow for writing. For both chips, the net effect of the instant stop is to eliminate from the seek time the average half-cycle shift of the minor loops.

The discussion in this section is not intended to imply that blocking is necessarily undesirable for MBMs. It does show that blocking is less important than for tapes, disks, and drums. It also indicates that blocking factors should be smaller for MBMs, and that there would be more occasions in which unblocked files, with less overhead, would be more efficient than blocked files.

## Random files

The small auxiliary storage application identified earlier for MBMs might very well benefit from file structures providing random access to any element in the file. Common structures providing this capability, other than hashing, are index or tree structures. The use of such structures is very dependent on the nature of the storage medium, and we now wish to analyze this use on MBMs.

A binary search tree is a very versatile structure for maintaining a file or table. It has excellent speed for searching, updating, and sequential processing.[11] Although it is not in general the fastest in any one of these activities, it can be the fastest for many applications requiring good performance in all three areas.

A generalization of the binary search tree is the multiway or $m$-way tree,[12] in which each node can contain up to $m$ sons and $m - 1$ keys, for some specified constant $m \geq 2$. Figure 5 illustrates a 4-way tree. A binary

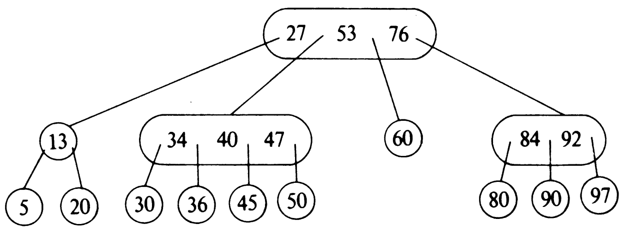

**Figure 5.** A 4-way tree.

search tree is a 2-way tree. A large value of $m$ is generally optimal for disk storage, because the number of accesses during a search operation is reduced. For main memory, $m = 2$ is optimal. Since MBMs lie between main memory and disks in terms of seek time, it is not obvious how best to structure multiway trees for them.

Assuming fixed length nodes, each node in an $m$-way tree must contain space for $m - 1$ keys and $m$ links. The nodes may contain the entire records corresponding to the keys, or the records may be stored in special nodes at the bottom of the tree. It may be desirable to have an additional field to indicate the number of keys in the node. We shall suppose each node has the form $[L_1, L_2, \ldots, L_m, R_1, R_2, \ldots, R_{m-1}]$, where $L_i$ is the $i$th link and $R_i$ is the $i$th record. We assume that the $i$th key is embedded in a known position of $R_i$.

Denote the size of each record by $s'$ and the size of the links by $p$. Let $s = s' + p$. Then the size of each node is $s(m - 1) + p$. Suppose there are $n$ records in the file, the average seek time to a node in auxiliary storage is $a$, and the transfer rate from auxiliary storage to main memory is $r$. Finally suppose that the time to search a node in main memory (using a binary search) is $b \log_2 m + c$, where $c$ is at most several microseconds.

The average time $t$ required to search for a random record in a balanced tree is then given by:[12]

$$\begin{aligned} t &= \log_m n[a + r(s(m-1) + p) + b \log_2 m + c] \\ &\simeq \log_m n[a + rs(m-1) + rp + b \log_2 m] \\ &\qquad\qquad (c \text{ small compared to } a) \\ &= \ln n[a/\ln m + rs(m-1)/\ln m + rp/\ln m + b/\ln 2] \quad (1) \end{aligned}$$

$t$ is minimized as a function of $m$ when $D_m(t) = 0$. We have

$$D_m(t) = \ln n/(\ln m)^2[-(a + rp)/m + rs(\ln m - (m-1)/m)]$$

Setting $D_m(t)$ to 0, we get

$$f(m) = m(\ln m - 1) = (a + rp)/(rs) - 1 = k \qquad (2)$$

Equation (2) involves the transcendental function $f(m) = m(\ln m - 1)$ and cannot be solved analytically. However, $f'(m) = \ln m$ which is positive for $m > 1$, therefore $f(m)$ is a monotonically increasing function for

$m > 1$. $f$ is obviously unbounded for $m > 1$ and $f(2) = 2(\ln 2 - 1) \simeq -0.614$, therefore for any $y \geq -0.614$ there exists a unique $x \geq 2$ such that $f(x) = y$. Equation (2) can easily be solved numerically for particular values of $a$, $r$, $s$ and $p$, thus leading to an optimal integer value of $m$ (if $k < -0.614$, $m = 2$ is optimal).

Table 2 gives the optimal tree order ($m$) for several different devices and record sizes. In each case, $p$ is assumed to be 4 bytes. Record sizes used were 8 bytes, 80 bytes, and 400 bytes, typifying very small, small (card image), and moderate record sizes. Recall that the record size includes the pointer size $p = 4$. The very small case of $s = 8$ bytes would very likely correspond to the situation in which only keys (and links) were stored in the internal nodes of the tree, and the complete records were stored in external nodes at the bottom of the tree.

**Table 2. Optimal order for multiway trees**

| Device | *a* (ms) | *r* (μs/bit) | *s* (bytes) | *k* | *m* | *g* (ms) |
|---|---|---|---|---|---|---|
| 1. Cartridge disk | 63 | 0.3 | 8 | 3281 | 607 | 11.6 |
| 2. Cartridge disk | 63 | 0.3 | 80 | 327 | 93 | 17.8 |
| 3. Cartridge disk | 63 | 0.3 | 400 | 64.6 | 28 | 26.7 |
| 4. Cartridge disk | 25 | 0.3 | 8 | 1302 | 281 | 5.4 |
| 5. Cartridge disk | 25 | 0.3 | 80 | 129 | 46 | 8.8 |
| 6. Cartridge disk | 25 | 0.3 | 400 | 25.0 | 15 | 14.2 |
| 7. Floppy disk | 360 | 6.0 | 8 | 937 | 214 | 82.4 |
| 8. Floppy disk | 360 | 6.0 | 80 | 93 | 36 | 138.0 |
| 9. Floppy disk | 360 | 6.0 | 400 | 17.8 | 12 | 229.9 |
| 10. Disk pack | 33 | 0.1 | 8 | 5156 | 890 | 5.7 |
| 11. Disk pack | 33 | 0.1 | 80 | 515 | 132 | 8.5 |
| 12. Disk pack | 33 | 0.1 | 400 | 102 | 39 | 12.3 |
| 13. Drum | 8.6 | 0.1 | 8 | 1343 | 288 | 1.8 |
| 14. Drum | 8.6 | 0.1 | 80 | 133 | 47 | 3.0 |
| 15. Drum | 8.6 | 0.1 | 400 | 25.9 | 15 | 4.8 |
| 16. MBM chip | 4.0 | 20 | 8 | 2.6 | 5 | 6.1 |
| 17. MBM chip | 4.0 | 20 | 80 | −0.6 | 2 | 25.2 |
| 18. MBM chip | 4.0 | 20 | 400 | −0.9 | 2 | 99.0 |
| 19. MBM chip | 7.3 | 10 | 8 | 10.9 | 9 | 5.8 |
| 20. MBM chip | 7.3 | 10 | 80 | 0.2 | 3 | 18.6 |
| 21. MBM chip | 7.3 | 10 | 400 | −0.8 | 2 | 57.2 |

The right-most column of Table 2, labeled $g$, gives values of the function $g(m) = a/\ln m + rs(m - 1)/\ln m + rp/\ln m$. Note that $t(m) = \ln n(g(m) + b/\ln 2)$, so that total access time (excluding internal searching) is $g(m) \ln n$. Cases 4, 5 and 6, which specify an average access time of 25 ms for a cartridge disk, assume that the file occupies a rather small number of cylinders (less than 10, say), and that arm movement is hence restricted to this subregion of the cartridge.

The most notable pattern in the table is that under comparable circumstances, the optimal order is much smaller for MBMs than for rotating disks and drums. The cause of this difference is clearly the faster seek times and slower transfer speeds of MBMs. For moderate sized records ($m = 400$) the optimal structure for MBMs is a binary tree, the same as for main memory. Even when the optimal order is 3, 4, or another small number, it may be preferable to use a binary tree owing to its greater flexibility regarding insertions, deletions, and balancing.

Insertions, deletions, and balancing comprise an important problem for multiway trees. An excellent solution to the problem is to use a special kind of multiway tree called a B-tree,[12,13] which guarantees a certain minimum balance and also facilitates insertions and deletions. A number of variations of B-trees are discussed in Refs 12 and 13, and the discussion is equally as applicable to MBMs as to rotating disks and drums.

The previous development in this paper for multiway trees largely applies also to B trees. The one important difference is the simplifying assumption in Eqn (1) that the tree is balanced. B-trees deal with the question of balance more precisely. In particular, it is shown in Ref. 12 that the number of accesses required to search a B-tree is less than or equal to $\log_{\lceil m/2 \rceil}((n + 1)/2)$. If $m > 2$ then $\log_{\lceil m/2 \rceil}((n + 1)/2) \leq \log_{m/2}((n + 1)/2) \cong (\ln(n + 1) - 0.7)/(\ln m - 0.7) = L$.

If $m$ (and $n$) is large, then $L \cong \ln n/\ln m = \log_m n$, which is the number of accesses used in Eqn (1).

If $m$ is small, a simple formula for $t(m)$ for a B-tree is not possible. Nevertheless, the same general conclusion would apply, namely that the optimal order for MBMs would be much smaller than for rotating disks and drums. The exact optimum for a particular situation could be determined through simulation.

## Indexed sequential files

Indexed sequential files attempt to provide a combination of the advantages of sequential processing and random processing. Three versions of indexed sequential files are IBM's ISAM (Indexed Sequential Access Method) and VSAM (Virtual Sequential Access Method),[10] and CDC's SCOPE.[14]

Indexed sequential structures are basically special kinds of tree structures in which the records themselves are stored as nearly as possible in physical sequential order. Insertions are accommodated by providing a certain amount of overflow space in the record nodes. Exceeding overflow space will result in some deviation from sequential storage.

IBM's ISAM is inherently based on rotating disk architecture, and attempts to reduce arm motion during sequential processing by storing logically adjacent records at least on the same cylinder if not physically adjacent. Such a consideration is not relevant to MBMs, since the seek component of access time is analogous to rotational delay on a disk, not arm movement.

Nevertheless, the basic features of the indexed sequential structure, a combination of indexing, sequential storage, and dispersed overflow areas, can be suitable for MBMs. Sequential processing on an MBM was discussed in a previous section, and the remarks also apply here. Similarly, the discussion of indexing in the previous section also applies to the indexing aspect of indexed sequential files. The new consideration, then, is for overflowing and the resultant possibility of nonsequential accessing of logically sequential records.

As noted, there is no need for concern about limiting arm motion on an MBM, as there is for a disk. When a nonsequential access must be made to the next record, however, it is advantageous if the access is just a little way down the minor loop from the present one. Since the MBM can stop instantly after each access, it would then need to seek only a short distance along the loop instead of halfway around on the average.

Unfortunately, if it is seeking to an overflow area, or a new area created out of sequence as a result of an overflow, then the seek back to the prime area would necessitate a rotation all the rest of the way around the loop, and nothing is gained. The only possible savings would occur if a sequence of two or more such nonsequential seeks could be spaced each a little further along than the previous one. Then moving from the original location, to the first overflow record, to the second overflow record, etc., could be done in one loop rotation.

We can look at this idea more carefully by considering the two common ways of handling the overflow problem in indexed sequential files, chaining and cellular splitting. ISAM reserves overflow areas in the form of overflow tracks or overflow cylinders. Records overflowing from a prime track are placed on the overflow tracks and/or overflow cylinders and are chained together in logically sequential order. If a chaining scheme such as this is used in bubble memory, then ideally it should be possible to move through every element in each chain in just one complete cycle of the minor loops. In this way, the time 'wasted' for sequential accessing of overflow records will be just one minor loop cycle per chain, instead of, say, one-half cycle per overflow record.

In order to achieve this property for the overflow chains without having to move overflow records, it will in general be necessary to store records in the overflow area according to a combination of a back-to-front pattern and a bisection pattern. The reason for the back-to-front pattern is that if the inserted record has a key which is smaller than the key of the last record on the primary track corresponding to the overflow chain, then that last record is bumped off the primary track and becomes the next overflow record. Its key will be less than all the other keys on the overflow chain, hence it should be stored in front of the other records in the overflow chain, hence the back-to-front storage pattern should be used.

This straight pattern, however, will not be suitable for handling cases in which the inserted record has a key which is higher than the key of the last record on the primary track but smaller than the key of the last record on the overflow track. In such cases the inserted record will itself be the next overflow record, and it will need to be stored after the logically preceding record in the chain and before the succeeding record in the chain.

Accordingly, it is desirable to leave gaps in the back-to-front storage pattern, to accommodate possible later insertions. The size of the gap should decrease as the length of the chain increases, since the probability of needing the gap would decrease (assuming 'random' insertions). Specifically, if $p$ is the number of records in the primary track (e.g. 20) and $r$ is the number of available overflow record cells in the overflow area in front of the first overflow record in the chain (e.g. 200), then a good value for the size of the gap is

$$G(p, r) = \frac{r}{p+2} \tag{3}$$

e.g. $200/22 \simeq 9$. For example, if the first record in the overflow chain is stored in overflow record position 274, then the new overflow record should be inserted in position 259, say, assuming six positions from 260 to 273 are taken (see Fig. 6).

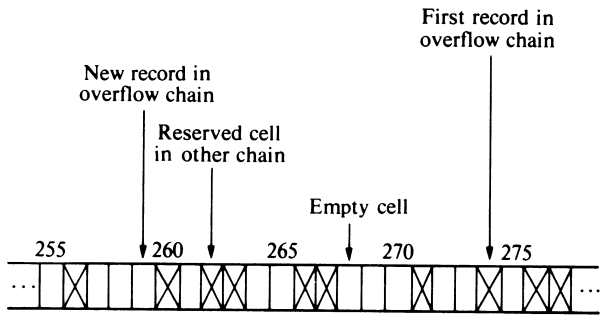
First record in overflow chain
New record in overflow chain
Reserved cell in other chain
Empty cell
255
260
265
270
275

**Figure 6.** Back-to-front storage allocation for ISAM overflow chain.

The rationale for this gap size is that after the insertion there will be $p + q$ primary and overflow records affiliated with the primary track, where $q$ is the number of records in the overflow chain. If there is a subsequent insertion into this set of records, the probability that it will fall between the first and second overflow records is $1/(p+q)$ (assuming equal likelihoods), whereas the probability that it will be somewhere before the first overflow record is $(p+1)/(p+q)$. The latter case will result in an insertion of a new overflow record prior to the current first one, hence the remaining $r$ available overflow cells should be apportioned according to the ratio $1/(p+q):(p+1)/(p+q)$, or $1:p+1$. This gives the fraction $r/(p+2)$ of Eqn (3). Note that $p$ is a constant whereas $r$ is decreasing (for a given chain). Note also that as a special case, if the overflow record is the first one in the chain then it should be stored in the backmost available overflow cell.

With this back-to-front pattern clarified, we still need to consider the case in which an insertion falls after the last primary record (technically, after the first overflow record). As noted earlier, it is then desirable to store the record in the gap between the preceding and succeeding records in the overflow chain. The best position to use in the gap would generally be assumed to be the middle position, so as to bisect the gap into two smaller gaps containing approximately equal numbers of available overflow cells. This policy yields the bisection pattern alluded to earlier.

It is possible, of course, that for either the back-to-front policy or the bisection policy, there may not be an available overflow cell which will maintain the overflow chain in the desired storage pattern. In other words, the 'gap' in which an insertion is to be made may have 0 length. In this case the insertion would simply be made elsewhere and the sequential access time would be degraded somewhat, or the overflow records would have to be moved. We shall not discuss any policy or algorithm for this situation.

It is apparent from the preceding discussion that there must be some scheme for keeping track of available cells in the overflow area. A bit map would be very suitable for this purpose. There is no advantage, incidentally, in having more than one overflow area, such as ISAM does. It is better instead to have one large contiguous area for overflow, with each prime track using this area for its overflow chains. Obviously the chains will intermingle, but this fact does not alter the previous analysis.

The original motivation for the storage allocation scheme presented for overflow records was to minimize the access time during sequential processing. The scheme also helps with random processing, however, since it will facilitate searching the overflow chain.

It should finally be noted, with regard to this ISAM discussion, that we have continued to use the basic ISAM organization. Specifically, we have assumed the use of structures analogous to primary tracks and overflow chains. Other organizations could have been used, of course, and that in fact brings us to the VSAM or SCOPE type of implementation of indexed sequential files.

VSAM and SCOPE use the technique of cellular splitting to help with the overflow problem. For specificity, we shall confine our attention to the VSAM organization. VSAM maintains overflow space in three different places: at the end of control intervals, at the end of control areas, and in a global area from which new control areas can be created.[10] If an insertion will fit in a control interval, the records are shifted to maintain physical sequential order, and there is no special problem with regard to MBM.

If a control interval overflows, it tries to split using a free control interval in the same control area. If this is not possible, the entire control area splits using a free control area from the global overflow space. In MBM the first type of cellular splitting is unnecessary, since shifting to the local overflow control interval, and then back to the next 'primary' control interval, will still waste one complete cycle of the minor loops. It is just as efficient (in access time) not to reserve any free control intervals in the primary control areas, but to use instead a single global free space for creating new control areas. Note that overflow space within a control interval is still beneficial in MBM. An additional advantage of a global free space is that it eliminates the possibility of 'wasting' free control intervals in control areas that have experienced few insertions, with the free intervals being unavailable to other control areas.

We shall conclude our discussion of indexed sequential files with a consideration of bidirectional shifting of the minor loops. The TIB 0103 and TIB 0303 do not have this capability, but it is a logical possibility for the major–minor loop architecture. Moreoever, it would have major implications for storage allocation for overflow records in indexed sequential files.

For both the chaining scheme and the cellular splitting scheme, local overflow space becomes worthwhile with bidirectional shifting. Using chaining, global overflow space should be dispersed uniformly around the minor loop, and an overflow record should be stored in the available cell which minimizes the sum of the distance from it to its predecessor and its successor. Using cellular splitting, free control intervals can be beneficial, and it is also desirable to disperse the global free space uniformly around the minor loop.

## Hashed files

Hashed file structures are quite dependent on the storage medium, and some analysis is appropriate before applying the structures to MBMs. Seek time, transfer rate, and memory size and cost are clearly important features in considering overflow rate, bucket capacity, probing sequence, etc. (cf. Ref. 12, pp. 506–540; Ref. 10, pp. 376–396). The general principles, rationale, and techniques developed for hashing still apply, but it is important to particularize the ideas to the performance characteristics of the new technology.

One special consideration regarding the implementation of hashed files in MBM concerns the use of bucketing, and the desirable size of the buckets. Table 3 gives the average number of accesses to secondary memory during a successful search to the file, as a function of the bucket size. A linear probe technique is assumed, and figures are given for both 80% and 90% packing densities. In general, the average number of accesses decreases as the size of the bucket increases, and this is the primary advantage of using large buckets. The primary disadvantage is that a large bucket requires a longer transfer time during an access.

**Table 3. Average accesses in a successful search using linear probing (cf. Ref. 12, pp. 506–540)**

| | Packing density | |
|---|---|---|
| Bucket size ($b$) | 80% | 90% |
| 1 | 3.000 | 5.500 |
| 2 | 1.903 | 3.147 |
| 3 | 1.554 | 2.378 |
| 4 | 1.386 | 2.000 |
| 5 | 1.289 | 1.777 |
| 10 | 1.110 | 1.345 |
| 20 | 1.036 | 1.144 |
| 50 | 1.005 | 1.040 |

Because MBM is so different from disks and drums in its seek time and transfer rate, the optimum bucket size is also radically different. To illustrate this, let $t$ denote the expected time for a retrieval (on either disk or bubble) and $a$ the average access time. We shall assume for simplicity that the average access time is the same for overflow probes as for the initial probe. In reality the time can be reduced using linear probing, but this fact does not alter the results which follow. Thus

$$t = a \times E[\text{number of accesses}]$$

Let us suppose that the disk has an average seek time of 33 ms (recall that we are including latency in the seek time) and a transfer time of 0.8 μs/byte. Suppose also that the MBM has an average seek time of 4 ms and a transfer time of 20 μs/byte. Let $s$ denote the size of the record in bytes, $b$ the number of records per bucket, and $t_d$ and $t_b$ the expected time required to retrieve a random record, using disk and bubble memory, respectively. Then

$$t_d = (33\text{ ms} + 0.8\text{ μs/byte} \times s \times b)E[\text{number of accesses}]$$

$$t_b = (4\text{ ms} + 20\text{ μs/byte} \times s \times b)E[\text{number of accesses}]$$

Table 4 gives the values of $t_d$ and $t_b$ for several bucket capacities and two record sizes, assuming an 80% packing density.

It is clear from the table that the optimal bucket size is much larger on disk than on MBM. This pattern holds in general for all record sizes. For example, for a record size of 300 bytes, the optimal bucket size is approximately 20 for the disk and still 1 for the MBM. The results hold in a general fashion for other similar seek and transfer

**Table 4. Expected retrieval time in a successful search using linear probing (80% packing density)**

| | $S = 500$ bytes | | $S = 100$ bytes | |
|---|---|---|---|---|
| $b$ | $t_d$ (ms) | $t_b$ (ms) | $t_d$ (ms) | $t_b$ (ms) |
| 1 | 100.20 | $\boxed{42.00}$ | 92.24 | 18.00 |
| 2 | 64.32 | 45.67 | 63.10 | $\boxed{15.22}$ |
| 3 | 53.15 | 52.84 | 51.65 | 15.54 |
| 4 | 47.96 | 60.98 | 46.18 | 16.63 |
| 5 | 45.12 | 69.61 | 43.05 | 18.05 |
| 10 | $\boxed{41.07}$ | 115.44 | 37.52 | 26.64 |
| 20 | 42.48 | 211.34 | $\boxed{35.85}$ | 45.58 |
| 50 | 53.27 | 506.52 | 37.19 | 104.52 |

speeds. If the bubble transfer rate were increased (through improved technology or the use of more chips in parallel), then it would become desirable to bucket with records larger than 300 bytes, but the bucket sizes would still be smaller than for disks. Finally, the general results also hold for unsuccessful searches and for other probing algorithms. It is interesting to note, though, that for an unsuccessful search using secondary hashing, the optimal bucket size is 1 (i.e. no bucketing) for MBM for records larger than just 40 bytes.

The nature of bucketing on MBM has some important implications for the handling of overflows and the probing sequence. For example, the 'consecutive spill' method (Ref. 10, pp. 376–396), in which overflow from one bucket causes a probe to the next consecutive bucket, benefits from a large bucket size. If the bucket size is small, as on MBM, then this method would not be very suitable. In general, MBM is surprisingly similar to main memory with regard to the overflow problem, especially if the bucket size is 1. Thus the advantages and disadvantages of collision-handling algorithms for main memory also usually apply to MBM.

## SUMMARY

We have described some typical MBM architectures and analyzed their access times. We have analyzed the effect of accessing records which are larger or smaller than the standard size. We have discussed the advantages and disadvantages of parallel and serial systems of MBM chips.

We have discussed a number of file structure considerations for the emerging MBM architectures, keeping in mind the most likely areas of application for the new devices. For the most part, the file structures discussed have already been applied to older technologies. In many significant instances, however, the nature or extent of such usage is substantially different from what it should be for MBMs, and we have tried to analyze these cases. This analysis has included sequential files, tree-structured files, indexed-sequential files, and hashed files.

## REFERENCES

1. H. Chang, Memory technology, magnetic bubble, in *Encyclopedia of Computer Science and Technology*, Vol. 10, ed. by J. Belzer, pp. 274–384. Marcel Dekker, New York (1978).
2. Electronics review, Bubble memory chips now able to pack in quarter million bits. *Electronics*, 39–40 (17 August 1978).
3. W. Myers, Current developments in magnetic bubble technology. *Computer*, 73–82 (August 1977).
4. J. E. Juliussen, Bubble memory as small mass storage, *Electro 77* (April 1977).
5. J. E. Juliussen, Magnetic bubble systems approach practical use. *Computer Design*, 81–91 (October 1976).
6. J. E. Juliussen, Bubbles and CCD memories—solid state mass storage. *National Computer Conference*, 1067–1075 (1978).
7. J. E. Juliussen, D. M. Lee and G. M. Cox, Bubbles appearing first as microprocessor mass storage. *Electronics*, 81–86 (4 August 1977).
8. C. K. Wong and P. C. Yue, Data organization in magnetic bubble lattice files. *IBM Journal of Research and Development*, 576–581 (November 1976).
9. W. Wright, External sorting using general purpose magnetic bubble memory. *Proc. COMPSAC 80*, 716–722 (1980).
10. J. Martin, *Computer Data-Base Organization*, 2nd Ed, pp. 332–396. Prentice-Hall, Englewood Cliffs, New Jersey (1977).
11. J. Nievergelt, Binary search trees and file organization. *Computing Surveys* **6** (3), 195–207 (1974).
12. D. E. Knuth, *The Art of Computer Programming*, Vol. 3, pp. 472–479. Addison-Wesley, Reading, Massachusetts (1973).
13. R. Bayer and E. McCreight, Organization and maintenance of large ordered indexes. *Acta Informatica* **1**, 173–189 (1972).
14. J. Tremblay and P. Sorenson, *An Introduction to Data Structures with Applications*, pp. 580–583. McGraw-Hill, New York (1976).
15. Texas Instruments, *TIB 0203 Magnetic-Bubble Memory System Application Manual*, Texas Instruments, Dallas (1979).
16. Texas Instruments, *Quarter-Million Bit Magnetic Bubble Memory*, Texas Instruments, Dallas (1979).
17. D. E. Knuth, *The Art of Computer Programming*, Vol. 1, pp. 435–451. Addison-Wesley, Reading, Massachusetts (1968).

Received March 1982