K. M. Chung
F. Luccio
C. K. Wong

On the Complexity of Permuting Records in Magnetic Bubble Memory Systems

In this paper we study the problem of permuting records in various simple models of magnetic bubble memories. Previous studies usually assumed the memory system either had one switch or n independently controlled switches, where n is the number of records to be permuted. In the former case, the time complexity to permute a set of n records is $O(n^2)$, while in the latter case, the time complexity is $O(n)$. In this paper, we propose several simple models of bubble memory systems with their numbers of switches ranging between 1 and n and analyze the respective time complexities and respective numbers of control states for some permutation algorithms designed especially for them. Specifically, four models are studied: They have essentially $\log_2 n$, $2\sqrt{n}$, $(\log_2 n - \log_2 \log_2 n)^2$, and k switches; their respective time complexities are essentially $(3/2)n \log_2 n$, $(5/2)n$, $(7/2)n$, and $2^{-1/k}kn^{1+(1/k)}$; and their respective numbers of control states are essentially $4 \log_2 n$, $2^{\sqrt{n}+1}$, $2n/\log_2 n$, and $4k$.

1. Introduction

The problem of permuting records in a magnetic bubble memory has recently received considerable attention [1-8]. Permuting records may constitute part of a sorting algorithm, in the sense that the keys of the records are sorted in the CPU and the records are then rearranged according to the sorted keys (hence a permutation of the records [3-6, 9]). Or the problem may arise from the following situation: Suppose the requests for the records do not occur randomly, and we accumulate statistics about their access frequencies. Then at certain time intervals we can rearrange the records according to these frequencies to improve upon the average access time. For example, the higher the frequency, the closer to the input/output port the record is placed [10-15]. In effect, a permutation of the records is performed.

Throughout this paper, we assume n is the number of records to be permuted. In the models of [3-6, 9] there are $(n - 1)$ independently controlled switches with time complexity $n/2$ for a permutation. Furthermore, the actual permutation time can be completely overlapped with the input/output time. On the other hand, the two models in [7], which are improvements of those in [8], have only one switch and the time complexities for permutation are both $(1/2)n^2 + O(n)$.

In this paper, we propose several models which require different numbers of independently controlled switches. For each model, we study the time complexity of a proposed permutation algorithm and the required number of control states. We assume a switch has two states. Thus, in general, a model with x switches may have a total of 2^x possible states. However, in our proposed algorithms only a subset of them is required. The cardinality of this subset is the number of control states necessary.

A summary of results is given in Section 7. The problem of sorting n records is treated in another paper [16].

2. Basic model

We assume that a magnetic bubble memory consists of *loops*. A loop of size m is capable of holding m records. Under the control of a switch, a loop can circulate records in a counterclockwise direction or can hold the rec-

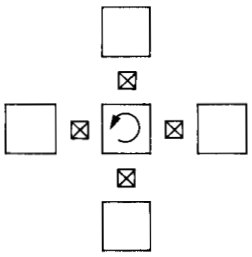

Figure 1 One major loop with four minor loops.

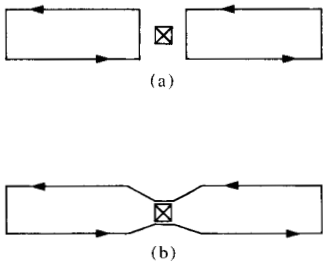

Figure 2 Adjacent loops cannot exchange records when switch is open as at (a) but can when it is closed as at (b).

ords in position. The time it takes to move one record from one position to an adjacent position (in a counterclockwise direction) is called a *step* and is assumed to be the basic unit of time. The "holding" action of the loops can be realized by one of the following two methods:

1. By placing an additional magnetic field over the loop, which, when turned on, cancels the original rotating magnetic field that drives the bubble domains around.
2. By using the model proposed in [1], which is an unconventional major/minor loop structure with specially designed switches (to be described in the next paragraph).

We use one minor loop for a record, with a switch connecting each of the minor loops to a major loop. Thus, a loop of size m in our model actually consists of one major loop and m minor loops. (See Fig. 1.) All the switches connecting the major loop to its minor loops are under one single control and can be opened or closed simultaneously. If they are all closed, in one step the bubbles in one minor loop move to an adjacent minor loop. On the other hand, if they are all open, then in one step the bubbles complete a cycle within the minor loops and the holding action is achieved.

We now describe the function of a switch for all our models. Two adjacent loops can exchange records by means of a switch. When the switch between them is open, the two loops remain separate. But when it is closed, they form a single big loop. (See Fig. 2.) These switches can also be used to implement the major-minor loop switches described in (2) above. Implementation of such switches in hardware has been demonstrated in many previous studies; see, *e.g.*, [9, 17].

3. Separation algorithm

In the models and algorithms to follow, one basic algorithm is always used; it separates the data into two adjacent loops connected by a switch. More specifically, given two loops of records with the destination loop of each record known, this algorithm describes a sequence of switch settings which moves the records to their destination loops.

Throughout this paper, we use the notation "move $(1, 2, \cdots, m;x)$" to mean "simultaneously shift loops $1, 2, \cdots, m$ by x steps." If $x = 1$, we simply write "move $(1, 2, \cdots, m)$."

```
procedure separate (i, j);
  (Comment: Perform a separation of records in adjacent
    loops i and j. Size [i] denotes the size of loop i, and
    destination [i, k] denotes the loop to which the record
    at position k of loop i is to be moved. Switch [i, j]
    designates the switch between loops i and j);
begin
  C1 ← 1;
  C2 ← 1;
  (Comment: Call the positions of the record of loop i and
    the record of loop j at the switch both 1);
  repeat
    if destination [i, 1] = i and destination [j, 1] = i then
      begin switch [i, j] ← "open";
        move (i);
        C1 ← C1 + 1;
      end
    else if destination [i, 1] = i
          and destination [j, 1] = j then
        begin switch [i, j] ← "open";
          move (i, j);
          C1 ← C1 + 1;
          C2 ← C2 + 1;
        end
    else if destination [i, 1] = j
          and destination [j, 1] = i then
        begin switch [i, j] ← "closed";
          move (i, j);
          C1 ← C1 + 1;
          C2 ← C2 + 1;
        end
```

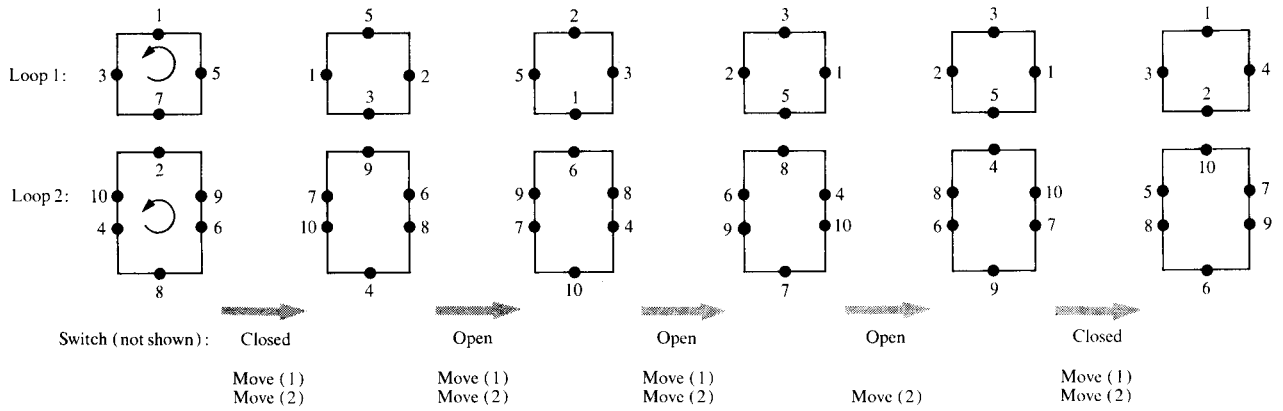

Figure 3 Sequence of switch settings required to separate lower-numbered records into loop 1 and higher-numbered records into loop 2.

```
    else if destination [i, 1] = j
          and destination [j, 1] = j then
        begin switch [i, j] ← "open";
          move (j);
          C2 ← C2 + 1;
        end;
  until C1 > size [i] and C2 > size [j];
end;
```

- *Example*

In the example of Fig. 3, there are two loops, designated 1 and 2. The size of loop 1 is 4 and the size of loop 2 is 6. A total of 10 records reside in these two loops. All records numbered less than or equal to 4 have loop 1 as their destination, while the others have loop 2 as their destination. At the beginning, records 7 and 2 are at the switch. The sequence of switch settings for the separation of records is "closed," "open," "open," "open," and "closed."

Note that, in general, the number of steps required by the separation algorithm is less than or equal to the sum of size [i] and size [j].

We next consider several models of magnetic bubble memories and propose corresponding permutation algorithms. All these algorithms have the previous separation algorithm as a basic building block. However, it is adapted to suit the specific models.

4. Model 1

In model 1, the bubble memory system consists of loops of sizes 2, 2, 4, 8 $\cdots$; *i.e.*,

$$\text{size } [i] = \begin{cases} 2^{i-1} & \text{for } i \geq 2, \\ 2 & \text{for } i = 1. \end{cases}$$

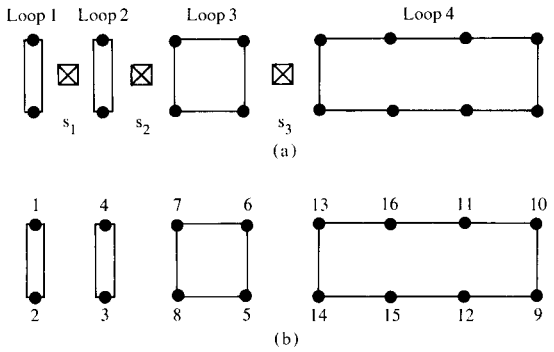

Figure 4 A 16-record memory is shown at (a) with its corresponding addresses at (b).

Thus, the total capacity of a memory system consisting of k loops is $2 + 2 + \cdots + 2^{k-1} = 2^k$. Between loop i and loop $i + 1$, for $i = 1, 2, \cdots, k - 1$, there is a switch s_i; thus, a total of $k - 1$ switches are required. (See Fig. 4.) Loops 1, 2, $\cdots$, $k - 1$ are regarded as forming the first half of the memory system, while loop k is regarded as forming the second half.

We impose a special ordering on the memory locations which we refer to as the memory addresses [Fig. 4(b)]. The permutation problem means that, for a given set of records with their relative ordering known beforehand, we want to place them in the memory locations such that the ith record goes to the ith memory location. The relative order of a record is also referred to as the address of the record.

Before stating the permutation algorithm formally, we first describe it informally:

1. The first step of the algorithm calls for the separation of records so that the ones with larger addresses go to the first half of the memory.
2. Then perform the desired permutation on them, while holding the records in the second half of the memory. The permutation of the records in the first half of the memory is done recursively.
3. Swap the first half with the second half by closing the appropriate switch (s_{k-1}) and circling the records through 2^{k-1} record lengths.
4. Repeat step 2.

This algorithm calls for a special address scheme for the memory locations. Assume the memory capacity is $n(=2^k)$ and regard the memory system as a $2 \times (n/2)$ array, with loop 1 occupying the first column, loop 2 the second column, loop 3 the third and loop 4 the fourth column, and so on. (See Fig. 4.) Thus, the first half of the memory occupies the first $n/4$ columns, while the second half of the memory occupies the remaining $n/4$ columns. Let (x, y), $1 \le x \le 2$, $1 \le y \le n/2$, be the coordinates of the memory locations and $d(x, y)$ the address to be assigned to (x, y). The address scheme is defined recursively as follows:

1. If $n = 2$, assign $d(1, 1) = 1$, $d(2, 1) = 2$.
2. If $n = 2^i$, for $i > 1$, then the address scheme for the first half of the memory is the same as when $n = 2^{i-1}$. For the second half of the memory, assign $d(x, y) = d(3 - x, (n/2) + 1 - y) + (n/2)$ for $1 \le x \le 2$, $1 + (n/4) \le y \le n/2$.

An example of this address scheme for a memory of capacity 16 is given in Fig. 4(b).

We can now describe the permutation algorithm formally. Note that in the sequel we write lg x for $\log_2 x$.

```
procedure swap (n):
  (Comment: Swap contents of the first (lg n − 1) loops
    with the (lg n)th loop by closing the switches and
    shifting the resulting big loop for half the total
    length);
begin
  for i ← 1 to (lg n − 1) do s_i ← "closed";
  (Comment: Set the first (lg n − 1) switches to
    "closed");
  move (1, 2, · · ·, lg n; n/2);
  (Comment: Move first lg n loops n/2 steps);
end;
```

The following procedure is an adaptation of the separation algorithm described in Section 3 to the current model.

```
procedure separate_1 (n, n_0);
  (Comment: Given n records with destination addresses
    n_0, n_0 + 1, · · ·, n_0 + n − 1 in the first lg n loops of the
    memory, the procedure performs a separation on the
    records such that the address of any record in the
    first (lg n − 1) loops is larger than that of any record
    in the (lg n)th loop. Denote by c(k) the destination
    address of the record currently at memory address k
    numbered as before);
begin
  for i ← 1 to (lg n) − 2 do s_i ← "closed";
  (Comment: The first lg n − 1 loops form a big loop);
  x ← (n/4 + 1);
  y ← (3n/4 + 1);
  (Comment: x and y are the memory addresses at the
    inputs to s_{lg n−1});
  count ← 0;
  for i ← 1 to n/2 do
    if c(i) < (n_0 + n/2) then count ← count + 1;
    (Comment: Count is the number of records that
      should be moved from the first (lg n − 1) loops to
      the (lg n)th loop);
  while count > 0 do
    if c(x) ≥ n_0 + n/2 and c(y) ≥ n_0 + n/2 then
      begin s_{lg n−1} ← "open";
        move (1, 2, · · ·, lg n − 1);
      end
    else if c(x) ≥ n_0 + n/2 and c(y) < n_0 + n/2 then
      begin s_{lg n−1} ← "open";
        move (1, 2, · · ·, lg n);
      end
    else if c(x) < n_0 + n/2 and c(y) ≥ n_0 + n/2 then
      begin s_{lg n−1} ← "closed";
        move (1, 2, · · ·, lg n);
        count ← count − 1;
      end
    else begin s_{lg n−1} ← "open";
        move (lg n);
      end;
end;
```

```
procedure permute_1(n, n_0);
  (Comment: Given n records with destination addresses
    n_0, n_0 + 1, · · ·, n_0 + n − 1 in the first lg n loops. Let
    c(i) denote the destination address of the record at
    memory address i. Given an initial set of c(i), which
    is a permutation of the set {n_0, n_0 + 1, · · ·, n_0 +
    n − 1}, the procedure performs a permutation of the
    records such that c(i) = n_0 + i − 1 for i = 1, · · ·, n);
```

```
begin
    if n = 2 then
      if c(1) > c(2) then move (1);
      (Comment: Put the records of loop 1 in order)
    else begin
      separate_1(n, n0);
      (Comment: Separate n records so that records with
        larger destination addresses go to the first half);
      s_{lg n-1} ← "open";
      permute_1 (n/2, n/2 + n0);
      (Comment: Apply recursion to the first (lg n − 1)
        loops);
        swap (n);
      s_{lg n-1} ← "open";
      permute_1(n/2, n0);
      (Comment: Apply recursion to the first (lg n − 1)
        loops while holding the (lg n)th loop);
    end;
end;
```

An example of permuting eight records is given in Fig. 5.

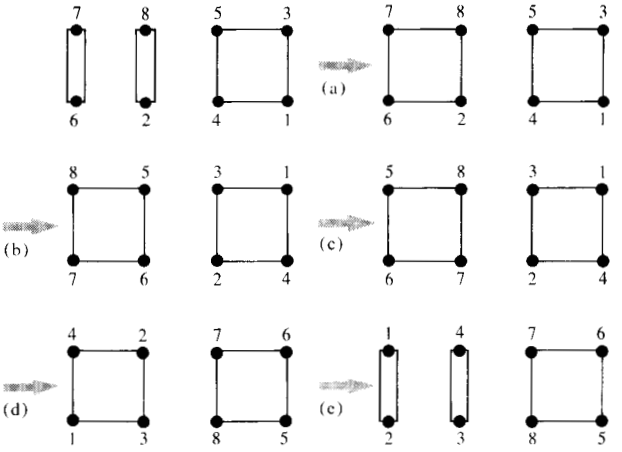

Figure 5 Eight records are permuted by first forming two big loops (a). Separation is carried out at (b), and the first half is permuted while the second half is held at (c). Loop contents are swapped (d), and the first half is permuted while the second half is held at (e).

We next analyze this algorithm. Let $p_1(n)$ be the number of steps to achieve an arbitrary permutation of n records. To separate a set of n records, the worst case for our separation algorithm requires n steps. (Actually, the worst case requires only $(n - 2)$ steps. This occurs when there are two loops of size $n/2$ each and the record destinations are as shown in Fig. 6. However, to simplify the calculation, we assume that the worst case for the separation algorithms is n). To swap two loops of size $n/2$ requires $n/2$ steps. Altogether we have

$$p_1(n) = n + p_1(n/2) + n/2 + p_1(n/2)$$

$$= \frac{3}{2} n + 2p_1(n/2)$$

$$= \frac{3}{2} n \lg n + c.$$

Since $p_1(2) = 1$, we have

$$p_1(n) = \frac{3}{2} n \lg n - 2.$$

The number of switches in the memory system is clearly

$$s_2(n) = \lg n - 1.$$

To compute the number of control states, we consider the situation when permutation is being performed on the first k loops, while loops $k + 1, k + 2, \cdots, \lg n$ are all on "hold." Four states are used in the procedure separate_1:

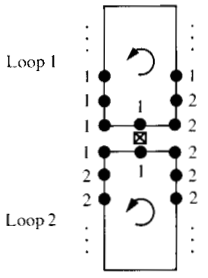

Figure 6 The number 1 adjacent to a record indicates that its destination is loop 1, while the number 2 indicates that its destination is loop 2.

1. Move first $(k - 1)$ loops, hold loop k;
2. Hold first $(k - 1)$ loops, move loop k;
3. Move all k loops, with switch s_{k-1} "open";
4. Move all k loops, with switch s_{k-1} "closed."

The procedure swap uses only the last state of the above. Therefore, for each step in the iteration, only four states are introduced. Thus, the total number of control states is

$$c_1(n) = 4 (\lg n - 1)$$

$$= 4 \lg n - 4.$$

5. Model 2

In model 2, the memory system consists of loops of equal size. After the data have been separated in the loops, the contents of the loops can be permuted *in parallel*.

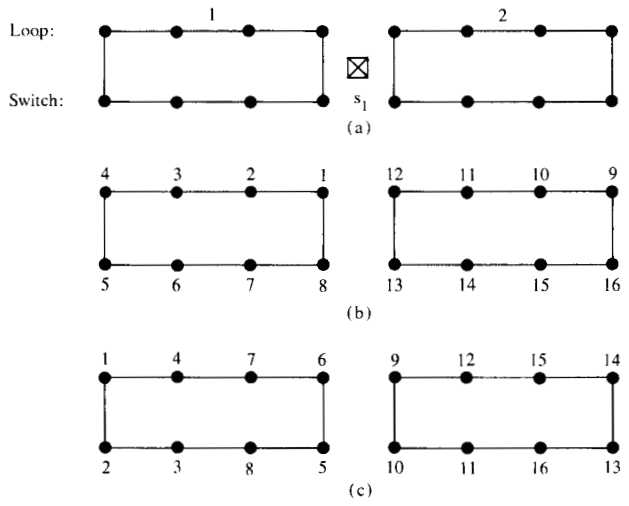

Figure 7 The memory at (a) has a capacity of 16. The corresponding addresses for model 2a are as shown at (b) and those for model 2b are as shown at (c).

In general, let the size of each loop be h and let the number of loops in the memory system be w. Between loops i and $i + 1$, there is a switch s_i, $i = 1, \cdots, w - 1$, with a total of $w - 1$ switches. [See Fig. 7(a).] We also assume that $w = 2^k$. Thus, the total capacity of the memory system is $n = 2^k h$. The final result depends on the structure of individual loops, which is described later.

The addresses of the memory locations are assigned sequentially from loop 1 to loop w, *i.e.*, loop 1 has memory addresses 1 to h, loop 2 has addresses $h + 1$ to $2h$, and so on. The memory addresses within the loops depend on the details of the loops and will become clear later. [See Figs. 7(b) and 7(c).]

The following procedure is an adaptation of the separation algorithm in Section 3 to the current model. Note that it differs from the adaptation for model 1 in that records with smaller addresses now go to the first half of the memory.

```
procedure separate_2(t, t_0);
  (Comment: Given t loops numbered t_0, t_0 + 1, ···, t_0 +
    t − 1, each of size h, and letting c(i) denote the destination address of the record currently at memory address i, the procedure separates the records of the t loops such that any record in the loops t_0, t_0 + 1, ···, t_0 + (t/2) − 1 has a destination address smaller than that of any record in the loops t_0 + (t/2), ···, t_0 + t − 1);
begin
    for i ← t_0 to t_0 + t/2 − 2 do s_i ← "closed";
    for i ← t_0 + t/2 to t_0 + t − 2 do s_i ← "closed";
    (Comment: Form two big loops);
    x ← memory address at the input of switch s_{t_0+t/2−1} in
      loop t_0 + t/2 − 1;
    y ← memory address at the other input of switch
      s_{t_0+t/2−1} in loop t_0 + t/2;
    a ← (t_0 + t/2 − 1) × h;
    for i ← 1 to t × h do
      if c(x) ≤ a and c(y) ≤ a then
        begin s_{t_0+t/2−1} ← "open";
          move (t_0, t_0 + 1, ···, t_0 + t/2 − 1);
        end
      else if c(x) ≤ a and c(y) > a then
        begin s_{t_0+t/2−1} ← "open";
          move (t_0, t_0 + 1, ···, t_0 + t − 1);
        end
      else if c(x) > a and c(y) ≤ a then
        begin s_{t_0+t/2−1} ← "closed";
          move (t_0, t_0 + 1, ···, t_0 + t − 1);
        end
      else s_{t_0+t/2−1} ← "open";
          move (t_0 + t/2, t_0 + t/2 + 1, ···, t_0 + t − 1);
      end;
end;

procedure permute_2(t, t_0);
  (Comment: Given t (assumed to be a power of 2) loops numbered t_0, t_0 + 1, ···, t_0 + t − 1, each of size h and letting c(i) denote the destination address of the record currently at memory address i, the procedure performs a permutation on the records such that c(i) = i);
begin if t = 1 then permute_x(h)
  (Comment: Procedure permute_x depends on the model used for a single loop)
    else begin
      separate_2(t, t_0);
      s_{t_0+t/2−1} ← "open";
      simultaneously begin
        permute_2(t/2, t_0);
        permute_2(t/2, t_0 + t/2);
      end;
    end;
end;
```

To analyze this algorithm, we let $p_2(n)$ be the number of steps to permute n records. Then

$$p_2(n) = n + p_2(n/2),$$

with initial condition

$$p_2(h) = p_x(h),$$

where $p_x(h)$ is the number of steps to permute the contents of a single loop and is dependent on the loop structure. Thus,

$$p_2(n) = 2n - 2h + p_x(h). \quad (1)$$

To specify the details of the individual loops, we have the following two models.

• *Model 2a*

Each loop has only one switch. (See Fig. 8.) When the switch is closed, the loop contents shift in a counterclockwise direction [Fig. 8(a)]. On the other hand, when the switch is open, the original loop is separated into two smaller loops [Fig. 8(b)]: one of size 1, the other of size $h - 1$. The records in the latter loop can shift in a counterclockwise direction independently of the single record loop. The memory addresses are assigned in a sequential fashion, starting with 1 for the location which becomes the loop of size 1 when the switch is open and continuing in a clockwise direction. Note that this is exactly the model 1 of [7]. By the permutation algorithm of [7], the number of steps to permute h records is at most $(1/2)h^2 + O(h)$. Thus, substituting in Eq. (1), we have

$$p_{2a}(n) = 2n + \frac{1}{2}h^2 + O(h). \quad (2)$$

If we choose $h = \sqrt{n}$, then

$$p_{2a}(n) = \frac{5}{2}n + O(\sqrt{n}),$$

and the total number of switches in the system is

$$s_{2a}(n) = 2\sqrt{n} - 1.$$

• *Model 2b*

Each loop is now structured exactly as in model 1 with the corresponding address scheme transplanted here as well. [See Fig. 7(c)]. Then,

$$p_{2b}(n) = 2n - 2h + \frac{3}{2}h \lg h - 2.$$

If we choose $h = 2^w$ (recall that w is the total number of loops in the system), then $n = hw = h \lg h$ and

$$p_{2b}(n) = \frac{7}{2}n - 2h - 2.$$

The total number of switches is

$$\begin{aligned} s_{2b}(n) &= (w - 1) + w(\lg h - 1) \\ &= w \lg h - 1 \\ &= \lg^2 h - 1. \end{aligned}$$

The fact that $n = h \lg h$ implies $h \sim (n/\lg n)$ for large n. Therefore,

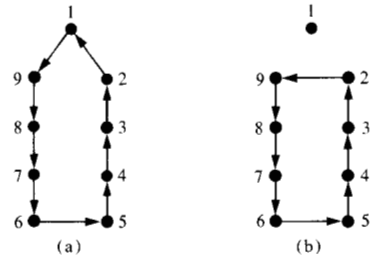

Figure 8 Model 2a with the switch closed (a) and open (b).

$$p_{2b}(n) \sim \frac{7}{2}n - \frac{2n}{\lg n} - 2,$$

and

$$s_{2b}(n) \sim (\lg n - \lg \lg n)^2$$

for large n.

To count the number of control states, we note that the procedure separate_2 requires only four states, as in model 1. With each iteration of the procedure permute_2, the operations in separate_2 are carried out in parallel, each independent of the other; hence, the total number of control states can be summed up as follows:

$$\begin{aligned} c_2(w) &= 4 + 4^2 + 4^4 + \cdots + 4^{w/2} + c_x(h, w) \\ &= \sum_{i=0}^{k-1} 4^{(2^i)} + c_x(h, w) \\ &= \sum_{i=1}^{k} 2^{(2^i)} + c_x(h, w), \end{aligned}$$

where $w = 2^k$, and $c_x(h, w)$ is the number of control states for the w individual loops and is included here to account for the control states used in the last recursion step. The value of $c_x(h, w)$ depends on the model used. Note that

$$c_2(w) < 2^{w+1} + c_x(h, w).$$

For model 2a, $w = \sqrt{n}$, $c_x(h, w) = 2^w$, since each loop has 2 control states; hence,

$$c_{2a}(n) < 3 \cdot 2^{\sqrt{n}}.$$

For model 2b, $w = \lg h$, to count the number of control states for the w individual loops, we note that each loop goes through $(\lg h - 1)$ iterations and each iteration is synchronized for all loops; *i.e.*, all loops go from one iteration to the other at exactly the same time. For any iteration, each loop requires 4 control states, with a total of 4^w control states. Thus, $c_x(h, w) = (\lg h - 1)4^w$ and

$$\begin{aligned} c_{2b}(n) &< 2^{\lg h + 1} + 4^{\lg h}(\lg h - 1) \\ &= (\lg h - 1)h^2 + 2h. \end{aligned}$$

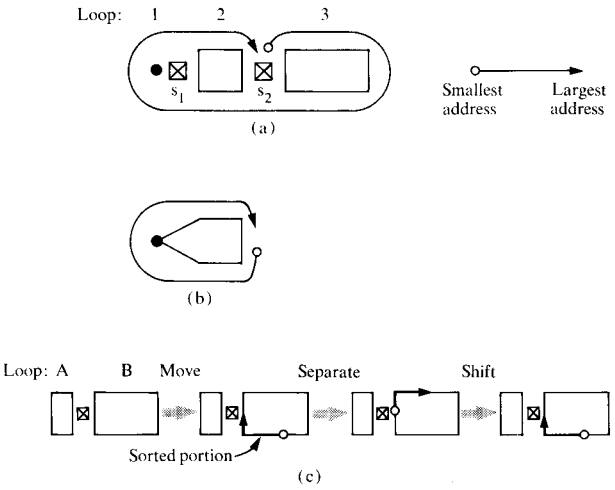

Figure 9 Algorithm for model 3: (a) address assignment; (b) desired permuted order in loops 1 and 2; (c) part 4 of the algorithm.

6. Model 3

In model 3, we assume that the number of switches is arbitrary but fixed, say, k. Suppose there are $k + 1$ loops of size $L_1, L_2, \cdots, L_{k+1}$, respectively, where $L_1 = 1$. Also assume there is a switch s_i between loop i and loop $i + 1$. Let $q_k(n; L_1, \cdots, L_{k+1})$ be the number of steps to achieve an arbitrary permutation of n records in this model.

For $k = 1$, using the algorithm in [7], we have $q_1(n; 1, n - 1) = (1/2)n^2 + O(n)$.

For $k = 2$, we have $n = L_1 + L_2 + L_3$. We next describe informally an algorithm which achieves an arbitrary permutation. This algorithm can then be recursively applied to larger k.

1. We assign the addresses in the three loops as in Fig. 9(a).
2. Regard loops 1 and 2 as one loop, loop 3 as another loop. For ease of discussion, we call them A, B, respectively. Separate the records into these two loops so that records with the smaller addresses are in loop A. (This part takes $L_1 + L_2 + L_3 = n$ steps.)
3. Permute the records in loop A so that the resulting order is as shown in Fig. 9(b), while holding the records in loop B. [This part takes $(1/2)(L_1 + L_2)^2 + O(L_1 + L_2)$ steps.]
4. Now move the sorted records from loop A to loop B. Then separate the remaining L_3 records into loops A and B so that records with the smallest addresses are in loop A. Next, shift loop B further so that the sorted records are in the same position as when they were first moved into loop B at the beginning of this step. [See Fig. 9(c).] (This part takes a total of n steps.)
5. Repeat parts 3 and 4 until all the addresses are in order. [This part is executed $\lceil L_3/(L_1 + L_2) \rceil$ times.]

The total number of steps required by this algorithm is therefore

$$q_2(n; L_1, L_2, L_3) = \left\lceil \frac{L_3}{L_1 + L_2} \right\rceil [n + q_1(L_1 + L_2; L_1, L_2)].$$

Since $q_1(L_1 + L_2; L_1, L_2) = (1/2)(L_1 + L_2)^2 + O(L_1 + L_2)$, it is clear that the best choice for L_2 is such that $L_1 + L_2 = a\sqrt{n}$ (this makes both terms n and q_1 have the same order of magnitude). Under this choice and omitting L_1, L_2, L_3 in the notation, we have [neglecting the term $O(L_1 + L_2)$]:

$$\begin{aligned} q_2(n) &= \left\lceil \frac{n - a\sqrt{n}}{a\sqrt{n}} \right\rceil \left(n + \frac{1}{2} a^2 n \right) \\ &< \frac{n}{a\sqrt{n}} \left(n + \frac{1}{2} a^2 n \right) \\ &= \left(\frac{1}{a} + \frac{a}{2} \right) n^{3/2}, \end{aligned}$$

since $\lceil x \rceil < x + 1$. The coefficient is minimum when $a = \sqrt{2}$. Thus we have

$$q_2(n) < \sqrt{2}\, n^{3/2}$$

and

$$L_2 = \sqrt{2n} - 1.$$

For $k > 2$, the procedure can be applied recursively. Suppose we can permute records on the first k loops using $(k - 1)$ switches. We let the first k loops form a single loop and loop $k + 1$ another loop. We then apply the same procedure as before; therefore,

$$q_k(n) = \left\lceil \frac{L_{k+1}}{\sum_{i=1}^{k} L_i} \right\rceil \left[n + q_{k-1}\left(\sum_{i=1}^{k} L_i \right) \right]. \tag{3}$$

Claim $q_k(n) < 2^{-1/k} k n^{1+1/k}$, if we assume

$$\sum_{i=1}^{k} L_i = 2^{1/k} n^{1-1/k}.$$

(Recall $\Sigma_{i=1}^{k+1} L_i = n$.)

Proof We prove this claim by induction. The claim is true for $k = 2$. From Eq. (3) and by the induction hypothesis, we have

$$\begin{aligned} q_k(n) &< \frac{n}{2^{1/k} n^{1-1/k}} [n + 2^{-1/(k-1)}(k - 1)(2^{1/k} n^{1-1/k})^{k/(k-1)}] \\ &= 2^{-1/k} n^{1/k} [n + (k - 1)n] \\ &= 2^{-1/k} k n^{1+1/k}. \end{aligned}$$

Note that the corresponding loop sizes $L_1, L_2, \cdots, L_{k+1}$ are, respectively,

$$1, (2^{1/2}n^{1/2} - 1), \cdots, (2^{1/k}n^{1-1/k} - 2^{1/(k-1)}n^{1-1/(k-1)}),$$
$$(n - 2^{1/k}n^{1-1/k}).$$

Remark When doing a permutation on two loops, our address assignment requires that the final configuration be as in Fig. 10(a). However, the recursive application of the algorithm requires that the final configuration be as in Fig. 10(b), so that it is ready to be moved out into the next bigger loop [the one on the right in Fig. 10(b)]. We can achieve this in the following way: As soon as the "head" of the record stream [*i.e.*, the record with the smallest address, *cf.* Fig. 9(c)] reaches the switch connecting the bigger loop [the rightmost switch in Fig. 10(b)], the switch is closed so that the record stream moves directly into the bigger loop [Fig. 10(c)] instead of circulating back. This modification in fact saves some running time.

To count the number of control states for model 3, we note that the process of separating records requires four states as before. Moreover, when the first p loops, say, are separating records, loops $p + 1, \cdots, k + 1$ are all on "hold." Also, two control states are sufficient to permute records in the first two loops. Therefore, the total number of control states is

$$c_3(n) = 4(k - 1) + 2$$
$$= 4k - 2.$$

7. Summary and conclusions

We summarize our results in Table 1; some known results are also listed for comparison. We also give some numerical examples in Table 2 to illustrate the relative values of these three parameters for the various models. We assume $n = 10^6$. Note that as far as the number of control lines is concerned, model 2a as well as the models in [3-6, 9] are practically not usable. Model 2b is of some theoretical interest since it gives a linear-time permutation algorithm with a small number of switches as well as control lines. It is also interesting to note that in model 3 the minimum number of steps is achieved when $k = \ln(n/2)$. Thus model 3 performs well only when k is small. (Note the substantial increase in performance when $k = 2$, $[O(n^{3/2})]$, as compared to the $O(n^2)$ performance in [7].) In terms of its simplicity in the permutation algorithm as well as the favorable values of its three parameters, model 1 seems to be a reasonable choice for a practical implementation of magnetic bubble memories.

The question still remains open as to the relationship of these three parameters in an optimal magnetic bubble memory designed for doing permutations of data. Except for the obvious fact that in a linear memory structure, where $O(n)$ steps are necessary to move a record from one end of the memory to the other, we know of no other theoretical lower bounds.

All models considered so far are essentially one-dimensional. It would be interesting to see if any two-dimensional memory structures would give better performance.

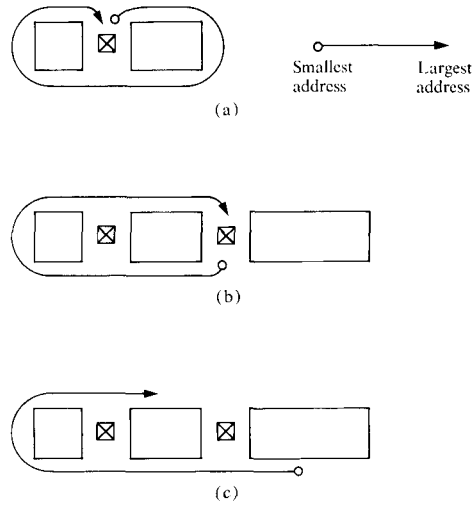

Figure 10 Our address assignment requires the final configuration at (a), but recursive application of the algorithm requires that at (b). With the configuration at (c), the record stream moves directly into the bigger loop instead of circulating back.

Table 1 Complexity of permuting records for different memory models.

Model	*Number of switches*	*Number of steps*	*Number of control states*
1	$\lg n - 1$	$\frac{3}{2} n \lg n - 2$	$4 \lg n - 4$
2a	$2\sqrt{n} - 1$	$\frac{5}{2} n + O(\sqrt{n})$	$<3 \times 2^{\sqrt{n}}$
2b	$\lg^2 h - 1$	$\frac{7}{2} n - 2h - 2$ ($n = h \lg h$; hence $h \sim n/\lg n$)	$< (\lg h - 1)h^2 + 2h$
3	k	$<2^{-1/k}kn^{1+1/k}$	$4k - 2$
[7]	1	$\frac{1}{2} n^2 + O(n)$	2
[3-6, 9]	$n - 1$	$\frac{n}{2}$	2^{n-1}

Table 2 Numerical examples of complexity.

Model	*Number of switches*	*Number of steps*	*Number of control states*	*Number of control lines* (=⌈lg (*control states*)⌉)
1	19	2.99×10^7	76	7
2a	2×10^3	2.5×10^6	1.07×10^{301}	10^3
2b	$(h = 6.27 \times 10^4)$ 254	3.5×10^6	1.25×10^5	17
3	$k = 2$	1.41×10^9	6	3
	$k = 3$	2.38×10^8	10	4
	$k = 4$	1.06×10^8	14	4
	$k = 12$	3.58×10^7	46	6
	$k = 13$ (lg $(n/2) = 13.1$)	3.57×10^7	48	6
	$k = 15$	3.60×10^7	58	6
[7]	1	5×10^{11}	2	1
[3-6, 9]	10^6	5×10^5	2^{10^6}	10^6

References

1. G. Bongiovanni and F. Luccio, "Permutation of Data Blocks in a Bubble Memory," *Commun. ACM* **22,** 21-25 (1979).
2. A. K. Chandra and C. K. Wong, "The Movement and Permutation of Columns in Magnetic Bubble Lattice Files," *IEEE Trans. Computers* **C-27,** 8-15 (1979).
3. T. C. Chen and C. Tung, "Storage Management Operations in Linked Uniform Shift Register Loops," *IBM J. Res. Develop.* **20,** 123-131 (1976).
4. T. C. Chen, K. P. Eswaran, V. Y. Lum, and C. Tung, "Simplified Odd-Even Sort Using Multiple Shift-Register Loops," *Research Report RJ 1919*, IBM Research laboratory, San Jose, CA, 1977.
5. T. C. Chen, V. Y. Lum, and C. Tung, "The Rebound Sorter: An Efficient Sort Engine for Large Files," *Research Report RJ 2204*, IBM Research laboratory, San Jose, CA, 1978.
6. F. Chin and K. S. Fok, "Fast Sorting Algorithms on Multiple Shift-Register Loops," *Technical Report*, Department of Computing Science, University of Alberta, Edmonton, 1977.
7. K. M. Chung, F. Luccio, and C. K. Wong, "A New Permutation Algorithm for Bubble Memories," *Research Report RC 7633*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1979.
8. C. K. Wong and D. Coppersmith, "The Generation of Permutations in Magnetic Bubble Memories," *IEEE Trans. Computers* **C-25,** 254-262 (1976).
9. C. Tung, T. C. Chen, and H. Chang, "A Bubble Ladder Structure for Information Processing," *IEEE Trans. Magnetics* **MAG-11,** 1163-1165 (1975).
10. W. F. Beausoleil, D. T. Brown, and B. E. Phelps, "Magnetic Bubble Memory Organization," *IBM J. Res. Develop.* **16,** 587-591 (1972).
11. P. P. Bergmans, "Minimizing Expected Travel Time on Geometrical Patterns by Optimal Probability Rearrangements," *Info. Control* **20,** 331-350 (1972).
12. P. I. Bonyhard and T. J. Nelson, "Dynamic Data Relocation in Bubble Memories," *Bell Syst. Tech J.* **52,** 307-317 (1973).
13. R. M. Karp, A. C. McKellar, and C. K. Wong, "Near-Optimal Solutions to a 2-Dimensional Placement Problem," *SIAM J. Computing* **4,** 271-286 (1975).
14. P. C. Yue and C. K. Wong, "On the Optimality of the Probability Ranking Scheme in Storage Applications," *J. ACM* **20,** 624-633 (1973).
15. P. C. Yue and C. K. Wong, "Near-Optimal Heuristics for an Assignment Problem in Mass Storage," *Intl. J. Computer Info. Sciences* **4,** 281-294 (1975).
16. K. M. Chung, F. Luccio, and C. K. Wong, "On the Complexity of Sorting in Magnetic Bubble Memory System," *Research Report RC 7634*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1979.
17. B. Antonini, M. Bacchi, and G. Bongiovanni, "A Bubble String Comparator for Information Processing," *IEEE Trans. Magnetics* **MAG-15,** 1183-1184 (1979).

Received March 6, 1979; revised May 25, 1979

K. M. Chung is at Wang Laboratories, One Industrial Avenue, Lowell, Massachusetts 01851, and F. Luccio is at the Istituto di Scienze dell'Informazione, University of Pisa, Italy. Part of the work was done while they were visitors at the IBM Thomas J. Watson Research Center. C. K. Wong is at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York 10598, but part of the work was done while he was a Visiting Professor at the Department of Electrical Engineering and Computer Science, Columbia University.