Parallel Sorting on a Re-circulating Systolic Sorter

F. S. Wong and M. R. Ito
Department of Electrical Engineering, The University of British Columbia, Vancouver, B.C., Canada V6T 1W5

In this paper, we present a re-circulating systolic sorting array and two sorting algorithms. The correctness of these algorithms is proved and general operational constraints are examined. These algorithms are amenable to VLSI implementation owing to the following attributes: (1) the simple control patterns of the algorithms, (2) the regular, repetitive and near-neighbour type of interconnection among the comparators, and (3) the systolic data movement. The sorting array is also well-suited for fabrication on shift-register type of storage and logic devices—such as magnetic bubble memories (MBMs) and charge-coupled devices (CCDs)—because of its closed-loop structure. The number of comparators and sorting time are both in $O(N)$ where N is the number of items to be sorted. A hardware termination method is incorporated into the control unit of the sorter, so that the sorting process can be terminated within a bounded time after the input list is in the desired order.

ABBREVIATIONS

N: Number of input items
C, Column # : Number of columns
R, Row # : Number of rows
P, Comparator # : Number of comparators
i: Loop index
j: Moving position index
J: Fixed position index
M_i: Initial marker's position in loop i
t: Comparison cycle time
"*": A marker

1. INTRODUCTION

Sorting is an important operation in business and computer engineering applications.[1] Many standard and novel sorting algorithms can be found in the literature,[1–9] some of them are optimal in time complexities, some in the number of comparators, whereas others lay emphasis on architectural designs—processor interconnections, data flow, control strategies and implementation technologies.

In this paper, we present a parallel sorting network which embodies the concept of systolic architectures.[2] Systolic systems are characterized by their data flow pattern: once data are loaded from the memories, they, and/or their intermediate results, circulate within the systems along predetermined paths provided among the processors, and every processor accepts and distributes data from and to its neighbours in a rhythmic fashion, analogous to the pulsations in the arteries caused by the recurrent contractions of the heart.

A major advantage of such systems lies in the fact that processor–memory communications are involved only during the loading of the input data and unloading of the final results. Therefore, there is no delay due to bus contention and memory fetch conflict during computation time.

A description of the systolic sorter and two examples are given in Section 2, and then two algorithms, two marking schemes and the constraints of the sorter are presented in Section 3. In Section 4, the algorithms are analysed and their timing complexities are discussed.

2. THE RE-CIRCULATING SYSTOLIC SORTER (RSS)

2.1 Network description

A schematic diagram of the proposed re-circulating systolic sorter (RSS) is given in Fig. 1.

The RSS network consists of an array of 'quadruple' comparators which are arranged into R rows and C columns. The whole array is articulated by $2R$ circular loops as shown. Each of the quadruple comparators holds and sorts four items during a comparison cycle, except those situated at the top and bottom of the array and located in the odd-numbered columns. Only the upper or lower portion of these comparators is used.

During the initial loading phase, all the loops are opened at the Input/Output switch and are connected to the input lines. Data items enter the network through the loops in a serial manner, with neighbouring loops shifted in opposite directions. When the network has been loaded, the input lines are disconnected and all the loops are closed. Before sorting commences, the comparator array has to be 'marked'. The purpose of the marking process is to place a marker in a certain position within each loop, to indicate the beginning and the end of that loop. The convention adopted here is that the markers are associated with the 'heads' of the loops, and the positions on the right-hand sides of the markers are the 'tails'. The readers may refer to the examples given in Figs 2(a) and 2(b), in which the markers are represented by asterisks. Note that the marking schemes—i.e. the ways to place the markers on the array prior to sorting—used in Figs 2(a) and 2(b) are different; they will be referred to as Scheme A and Scheme B, respectively.

After the marking procedure, one of the RSS algorithms will be applied to the array. During a comparison cycle, data items are compared and exchanged within the quadruple comparators; if a pair of items has to be exchanged, their associated markers, if there are any, do

CCC-0010-4620/84/0027-0260$05.00

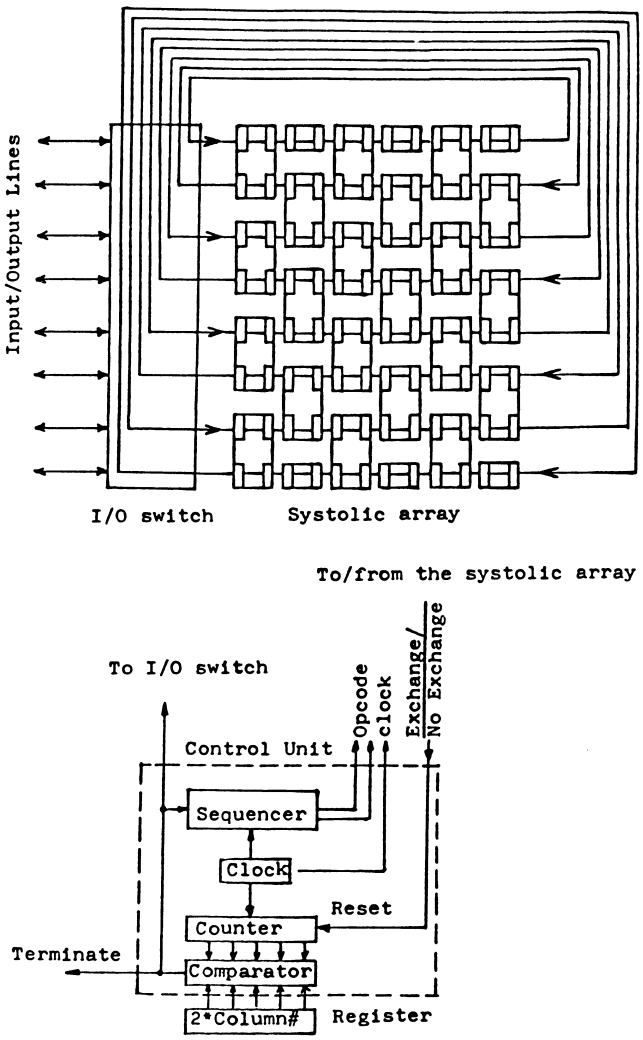

Figure 1. The re-circulating systolic sorter with its control unit.

not move with them; instead they will remain where thay are. However, between successive comparison cycles, when the items are shifted, the markers will be shifted along with the items.

A schematic diagram of the control unit used is also presented in Fig. 1. This unit generates the control signals ('Opcode' in Fig. 1) to indicate one of the operations to be performed by the comparators: (1) vertical comparison, (2) horizontal comparison, (3) diagonal comparison and (4) shift operation. At the end of each comparison cycle, the control unit also tests the status of the array ('exchange/no exchange') to see whether any exchange has taken place during that cycle. It also has a cycle counter, which keeps track of the current number of consecutive 'no exchange' cycles. In other words, the content of the counter is incremented upon entering a new cycle, and is reset whenever there is at least one exchange in that cycle. When the count reaches twice the number of columns (i.e. count $= 2C$), a termination signal will be generated. At this stage, the input items have been sorted into a linear list. As demonstrated in Figs 2(a) and 2(b), the first item of the sorted list is accompanied by an asterisk in the uppermost loop, and the last item is on the right-hand side of the asterisk in the lowest loop.

2.2 The quadruple comparator

The quadruple comparators have a higher logic density than the binary comparators used in other sorting networks, but the number of input/output lines per comparator of the former is only slightly more than that of the latter. Figure 3 gives a sketch of the input/output configuration of a quadruple comparator.

In addition to the two sets of input and two sets of output data lines, there are four single-bit lines used for shifting of markers along the two loops connected to the comparator, one line for clock signal, one line to indicate whether any exchange has taken place during the current comparison cycle, and two lines for the opcodes.

Essentially what a comparator unit accomplishes is the following. If it is located in an odd column, it pushes the smallest of the four items which it holds to its upper right neighbour, and the largest to its lower left neighbour. If it is in an even column, it retains the smallest and the largest items in its upper right and lower left positions, respectively. However, when markers are present inside the comparator, the situation becomes somewhat different and will be described in Section 3.1.

2.3 Examples

Two examples with three columns ($C := 3$), three rows ($R := 3$) and eight comparators ($P := 8$) are presented in Figs 2(a) and 2(b). After the initial loading and marking procedures, Algorithms I and II are applied to the examples in Figs 2(a) and 2(b), respectively. The contents of the comparator array are shown for the first and the last two cycles. Both input lists are sorted into ascending order. At the end of the last cycle, the minimum of each loop is indicated by the marker, and the direction of increasing values is from right to left. All the numbers in a given loop are more positive than or equal to those in the next loop above.

3. THE RSS ALGORITHMS

For the convenience of illustration, the symbols shown in Fig. 4 will be used throughout this paper.

The direction of comparison is used to show the ordering of items after each comparison. In Fig. 4(a), the solid arrow head indicates the position of the larger item for ascending order. If, on the other hand, descending order is desired, then the arrow head indicates the smaller item. Without loss of generality, ascending order will be assumed in this paper. The open arrow of Fig. 4(b) is used to indicate the direction of movement for both the items and the markers during the shift operations.

3.1 The comparison/exchange/shift operations

The four operations performed by a comparator are depicted in Fig. 5 and are described below:

1. Vertical comparison: the two items on the upper portion of the comparator are compared to the two at the bottom in parallel, with the directions of comparison pointing downward. The presence of the marker is ignored.

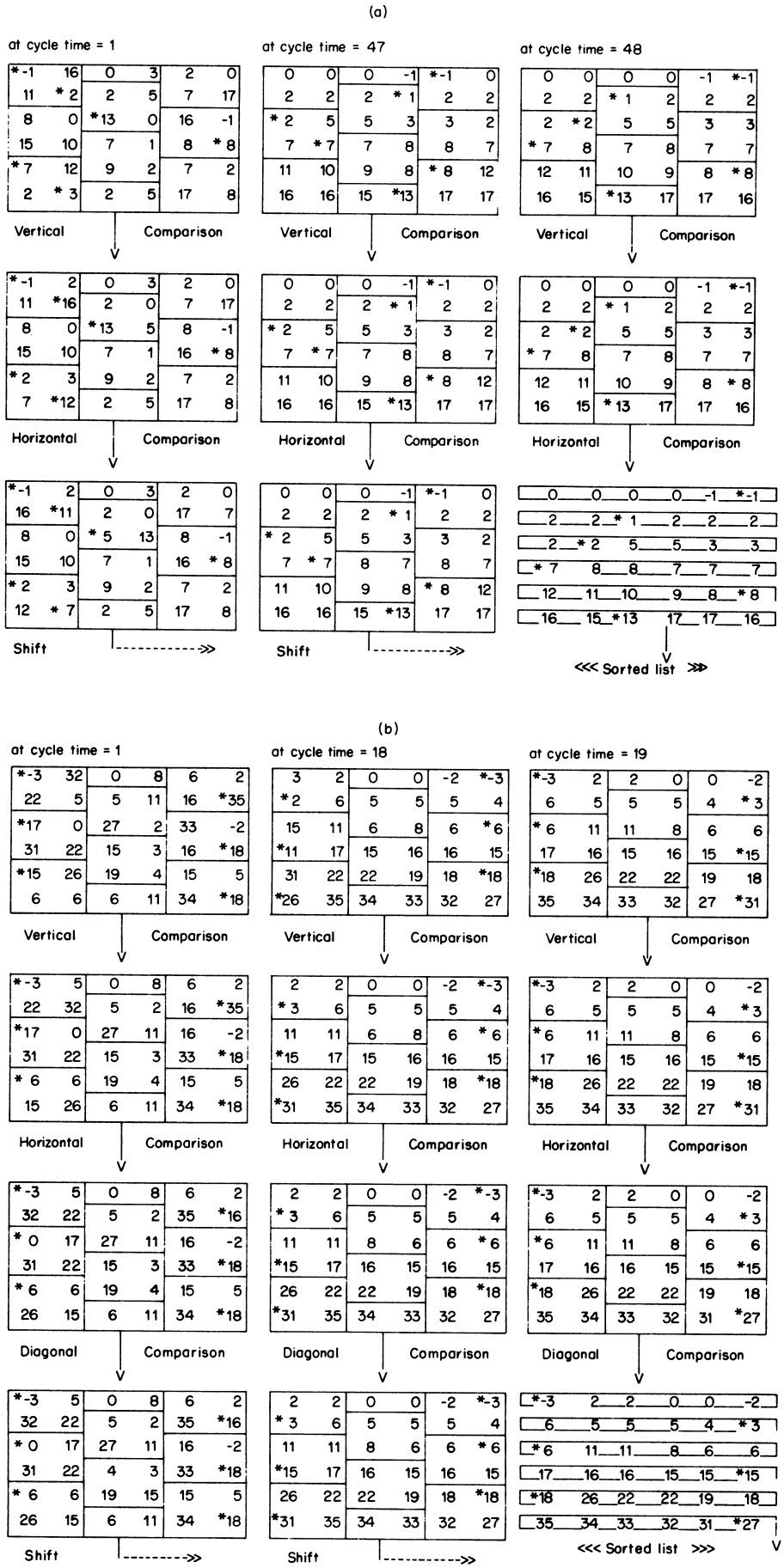

Figure 2. (a) An example to illustrate RSS Algorithm I, (b) An example to illustrate RSS Algorithm II.

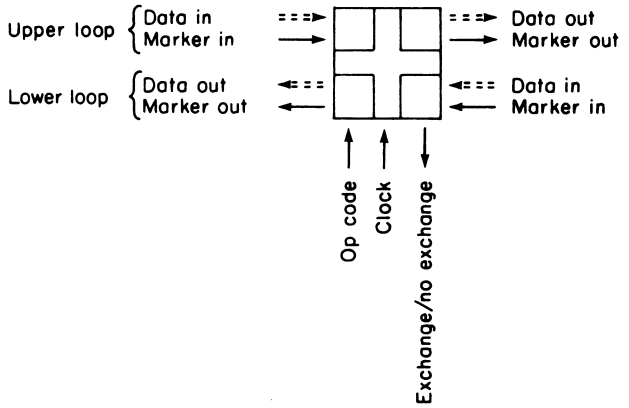

Figure 3. The schematic diagram of a quadruple comparator unit.

2. Horizontal comparison:

   Case (*i*). No marker is inside the comparator: the two items on the right portion of the comparator are compared to the two on the left in parallel, with the directions of comparison pointing to the left.
   Case (*ii*). One or two markers are inside the comparator: when a marker appears on the left portion of the comparator, the corresponding direction of comparison points to the right; otherwise it points to the left.

Note that in the horizontal comparisons, the direction of comparison always points from right to left, unless both the head and tail of a loop are involved in the comparison—i.e. when the marker appears on the left portion of the comparator, then the direction is reversed. This reversal prevents the minimum and maximum items in a loop from crossing over each other, and it is achieved by the actions taken in Case (ii) above.

3. Diagonal comparison: the two items on the upper portion of the comparator are compared to the two at the bottom in parallel, with the directions of comparison pointing downward and crossing each other.

   At first glance, the diagonal comparison involving the top right and lower left items seems redundant, because these two items are already in order after the vertical and the horizontal comparisons. However, it is useful when two markers appear on the left portion of the comparator simultaneously. Furthermore, the top-left and bottom-right comparison provides an exchange not provided by the combination of the vertical and the horizontal comparisons.

4. Shift:

   Case (*i*). If a comparator is located in an even column, its top two items are shifted to the left, and its lower two items are shifted to the right.
   Case (*ii*). If a comparator is located in an odd column, its top two items are shifted to the right and its lower two items to the left.

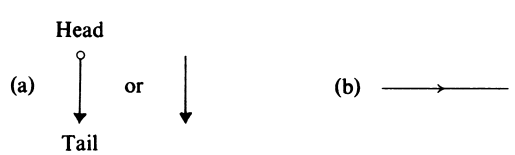

Figure 4. The symbols used. (a) Direction of comparison. (b) Direction of shift.

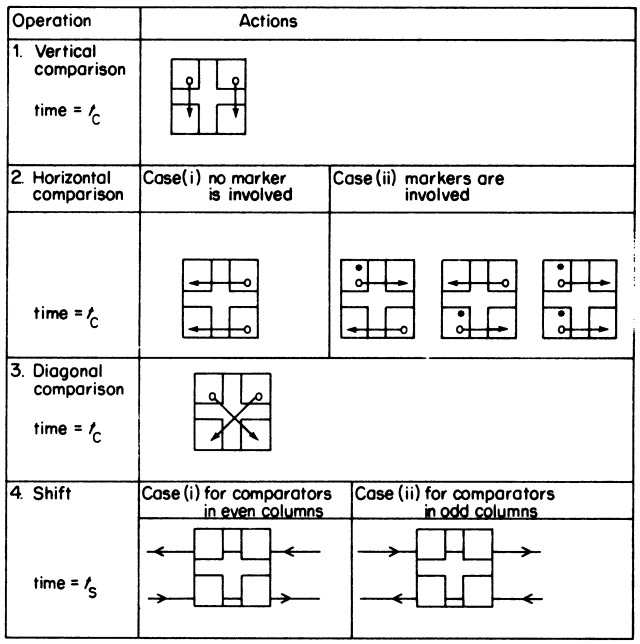

Figure 5. The 4 operations performed by the quadruple comparators.

3.2 Algorithm I

This algorithm involves only operations 1, 2 and 4, and is described in the following program fragment written in Pascal:

```
Program Systolic_Sorter:
  :
  :
Var
  Terminate      : boolean;
  Column #       : integer;
  Row #          : integer;
  Comparator #   : integer;
  Exchange       : boolean;
  (*to indicate whether any exchange has taken place
    during a comparison cycle*)
  Count_no_Exchange : integer;
  (*to count number of consecutive cycles with no
    exchange*)
    :
    (*Initialization*)
    :
    While NOT Terminate do
    (*enter next cycle of comparison*)
    Begin
      for C := 1 to Comparator # do
        Begin
          Vertical_Comparison;
          Horizontal_Comparison;
        End;
      Check_Terminate;
      Shift;
    End;
    :
    :
```

The procedure *Check_Terminate* manipulates the following global variables:

1. *Exchange*—This Boolean variable is always reset to be *False* before the next comparison cycle commences, and is set to be *True* if any exchange takes place during that cycle.
2. *Count_No_Exchange*—This variable counts the number of consecutive cycles which have no exchange and is reset to zero whenever *Exchange = True*.
3. *Terminate*—This Boolean variable controls the **while-do** loop, and it is set to be '*True*' if the following condition is satisfied:

Condition 1 (for termination).

$$Count\_No\_Exchange > 2 * Column\ \#$$

3.3 Algorithm II

This algorithm is similar to Algorithm I except that the *Diagonal Comparison* operation is included in its **while-do** loop:

```
Program Systolic_Sorter;
  :
  :
  While NOT Terminate do
  (*enter next cycle of comparison*)
  Begin
    for I := 1 to Comparator # do
      Begin
        Vertical_Comparison;
        Horizontal_Comparison;
        Diagonal_Comparison;
        (*for Algorithm II only*)
      End;
    Check_Terminate;
    Shift;
  End;
```

3.4 Constraints on the dimension of the RSS

Most sorting networks impose certain constraints on the dimension of the networks. For examples, the Batcher's bitonic sorter[1] requires that the number of its input lines be a power of two, and some mesh sorters[6,9] work on square arrays only. The basic constraint of the RSS array appears to be less stringent:

Requirement (1).

$$Column\ \# \geq 2$$
$$Row\ \# \geq 1$$

Further constraints may or may not be required depending on the marking schemes used. In Schemes A and B described below, Requirement (1) is sufficient to guarantee correct operation of both RSS algorithms when Scheme A is used, but an additional constraint on the dimension of the RSS array is imposed when Scheme B is used.

3.5 Marking scheme A

It is observed that only certain ways of making the array can guarantee correct results, and one such way is as follows:

The initial marker position, M_i, of loop i is given as

$$M_i := f(i) - M_{i-1}, \text{ for } i := 1, 2, \ldots, 2 * Row\ \# - 1$$

when i = odd,

$$f(i) := 4 * (\text{An odd integer}) - 2 \pm 1$$
$$\in \{3, 9, 11, 17, \ldots\}$$

when i = even,

$$f(i) := 4 * (\text{An even integer}) - 2 \pm 1$$
$$\in \{5, 7, 13, 15, \ldots\}$$

where $1 \leq M_i \leq 2 * Column\ \#$, and $M_{i=0}$ can be any value in the range of M_i.

Scheme A is adopted in the example given in Fig. 2(a), where $M_0 = 1$, $M_1 = f(1) - M_0 = 3 - 1 = 2$, $M_2 = f(2) - M_1 = 5 - 2 = 3$, $M_3 = f(3) - M_2 = 9 - 3 = 6$, $M_4 = f(4) - M_3 = 7 - 6 = 1$, $M_5 = f(5) - M_4 = 3 - 1 = 2$ and the pattern repeats. If there are only two columns, then $M_3 = 6$ MOD $2 * Column\ \# = 6$ MOD $4 = 2$. The rationale behind this scheme will be given in Section 4.2.

3.6 Marking scheme B

In this second scheme, the markers are placed along the two sides of the comparator array, as demonstrated in Fig. 2(b). This method is simpler and we can use the I/O lines to insert the markers, and also the retrieval of the final sorted list is easier than that of Scheme A.

This scheme requires that the number of columns of the RSS array be twice an odd integer, or the next higher integer to that value.

$$M_i := \begin{cases} 1 & , \text{for } i = \text{even} \\ 2 * Column\ \# & , \text{for } i = \text{odd} \end{cases}$$

Requirement (2). (For Scheme B only)

$$Column\ \# := \begin{cases} 2 * \text{An odd integer, or} \\ 2 * \text{An odd integer} + 1 \end{cases}$$
$$\in \{2, 3, 6, 7, \ldots\}$$

4. ANALYSIS OF THE RSS ALGORITHMS

4.1 Analogy with the odd–even transposition sort

The RSS algorithms bear some resemblance to the odd–even transposition sort,[4] therefore, a simple explanation of the odd–even sorter would be helpful in analysing the RSS network.

In Fig. 6, the appearance of an arrow indicates the presence of a binary comparator located at that position.

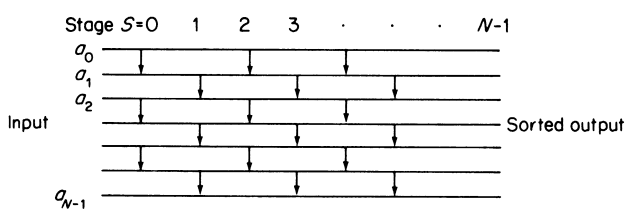

Figure 6. The odd–even sorter.

An item a_j is compared to another item $a_{j'}$ at stage s if,

$$j' = j + (-1)^{j+s} \quad (1)$$

where $0 \le j, j', s \le N-1$, and N is the size of the input list.

The value of j' alternates between $(j+1)$ and $(j-1)$ when s is incremented. This sorter guarantees sorting of N items in N cycles, but it requires a total of $N(N-1)/2$ comparators;[4] it is therefore impractical if N is too large.

4.2 Correctness of the RSS algorithms and the marking schemes

This section will prove that the RSS algorithms are correct by showing that the odd–even sort is actually performed on the RSS array. Let us consider three indices, i, j and J on the systolic array. As demonstrated in Fig. 7, $i(=0, 1, \ldots, 2R-1)$ indexes the loops, $J(=1, 2, \ldots, 2C)$ indexes the fixed positions of the array, and for each loop i, M_i indicates the initial position of the marker of the loop, and $j(=0, 1, \ldots, 2R-1)$ indicates the distance of a position away from the marker. Because the markers are shifted with time, j is a function of time and it is related to other indices as follows:

$$j := [(M_i + 2C - J) + (2C + (-1)^i t \text{ MOD } 2C)] \text{ MOD } 2C \quad (2')$$

$$= [4C + M_i - J + (-1)^i t \text{ MOD } 2C] \text{ MOD } 2C \quad (2)$$

The first composite term in (2′) shows the effect of the initial marker's position on j, and the second term is due to the effect of time indexed by t. The Modulo function is used to trim the values of t and j because both of them are repetitive with period $2C$. The readers can verify that expression (2) is correct for the example given in Fig. 7.

Having established the relationship among the indices, we will now derive several expressions to relate a pair of items $(a_{i,j} - a_{i',j'})$ involved in a comparison. First, let us consider the horizontal comparison.

In Fig. 7, $a_{i,J}$ is always compared to $a_{i,J'}$ where

$$J' := J - (-1)^J \quad (3)$$

Converting J and J' into j and j' using (2) and (3), we have

$$j = [4C + M_i - J + (-1)^i t \text{ MOD } 2C] \text{ MOD } 2C$$

$$j' = [4C + M_{i'} - J' + (-1)^{i'} t \text{ MOD } 2C] \text{ MOD } 2C$$

$$= [4C + M_{i'} - J + (-1)^J + (-1)^{i'} t \text{ MOD } 2C] \text{ MOD } 2C$$

In a horizontal comparison, $i = i'$; therefore

$$j' = j + (-1)^J \quad (3')$$

Again from (2),

$$J = j + 4C + M_i + (-1)^i t \text{ MOD } 2C + K * 2C$$

where K is any positive integer.

Substituting J into (3′), we obtain the expression for j', where $a_{i',j'}$ is compared to $a_{i,j}$ horizontally,

$$j' := j + (-1)^{j + M_i + 2C + (-1)^i t \text{ MOD } 2C} \quad (4)$$

Within loop i, M_i and $(-1)^i$ are constants, therefore j' will alternate between $(j+1)$ and $(j-1)$ as t increases. By comparing (4) and (1), we can see that the horizontal comparisons coupled with the shifting actions are equivalent to the odd–even sort as far as loop i is concerned, and the items within a loop can be sorted with $2C$ cycles. Figure 8 further demonstrates this point.

In addition to the near-neighbour comparisons as in the odd–even sort, the RSS also compares the head and tail of every loop, i.e. $(a_{i,0} - a_{i,2C-1})$. These comparisons do not upset the sorting process because they are taken care by case (ii) of horizontal comparison described in Section 3.1. The whole input list $(a_{0,0}\ a_{0,1} \ldots a_{0,2C-1})$ $(a_{1,0}\ a_{1,1} \ldots a_{1,2C-1}) \ldots (a_{2R-1,0}\ a_{2R-1,1} \ldots a_{2R-1,2C-1})$ can be sorted using the same principle provided

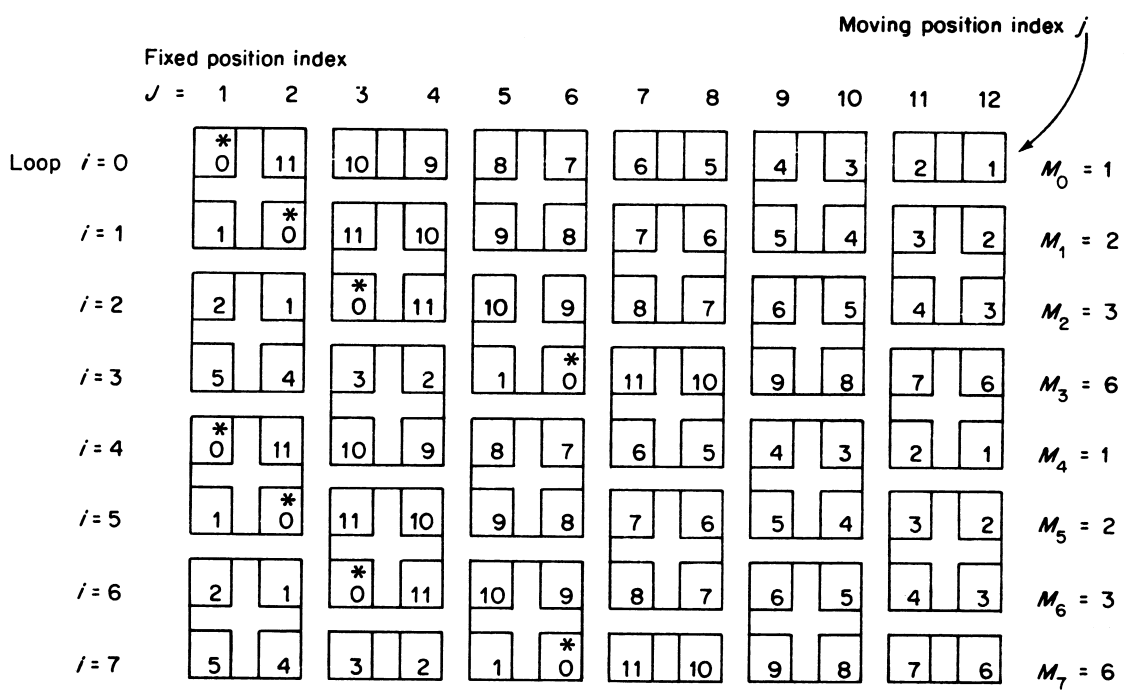

Figure 7. The three indices i, j and J, and the initial marker position M_i.

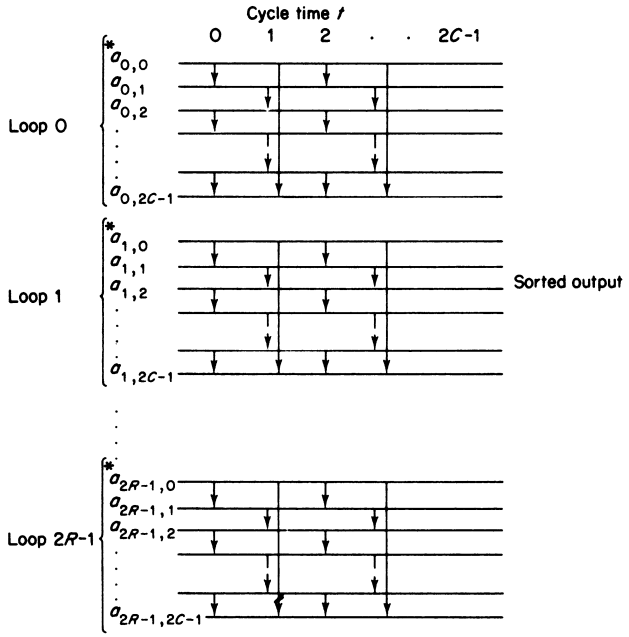

Figure 8. The horizontal comparisons carried out on the RSS array.

that the tail of each loop is compared to the head of the next lower loop, i.e.

$$(a_{i,j=2C-1} - a_{i'=i+1,j'=0}), \quad \text{where } i = 0, 1, 2, \ldots, 2R-1 \quad (5)$$

These comparisons are carried out by either the vertical comparison or the diagonal comparison operations. Let us consider the vertical comparisons between a pair of items $(a_{i,j} - a_{i',j'})$. From Fig. 7, note that $a_{i,J}$ is always compared to $a_{i',J}$ if,

$$i' = i - (-1)^{i+\lceil J/2 \rceil} \quad (6)$$

From (5), (6) and (2), we can obtain the position J where the heads and tails of the loops meet:

$$j = 2C-1, \Rightarrow 4C + M_i - J + (-1)^i t \text{ MOD } 2C = K*2C + 2C - 1 \quad (7)$$

$$j' = 0, \Rightarrow 4C + M_{i'} - J + (-1)^{i+1} t \text{ MOD } 2C = K'*2C \quad (8)$$

Combining (7) and (8),

$$8C + M_i + M_{i'} - 2J = (K+K')*2C + 2C - 1$$
$$\Rightarrow J = K''C + \frac{M_i + M_{i'} + 1}{2} \quad (9)$$

where K and K' are integers such that $0 \le j \le 2C-1$, and K'' equals either -1, 0 or 1 because $1 \le J \le 2C$. Expression (9) means that the tail of loop i will be compared to the head of loop $i+1$ at either halfway between M_i and M_{i+1}, i.e. $J = (M_i + M_{i+1} + 1)/2$, or $J = (M_i + M_{i+1} + 1)/2 \pm C$ depending on whether there is any quadruple comparator situated at these locations. From (9),

$$M_i + M_{i+1} + 1 = 2(J - K''C)$$
$$\Rightarrow M_i + M_{i+1} = 2(J - K''C) - 1 \quad (10)$$

this gives rise to another requirement for the marking of the comparator array:

Requirement (3). (For marking schemes)

$$M_i + M_{i+1} = \text{An odd integer}$$

This requirement will ensure that the tails and heads of the loops will be compared by the vertical comparisons. It is automatically satisfied when marking schemes A or B are used; however, it has to be considered if other marking schemes are used.

Now we will derive marking scheme A. From (5) and (6), we have

$$i' = i+1 \text{ and } i' = i - (-1)^{i+\lceil J/2 \rceil}$$
$$\Rightarrow i + \lceil J/2 \rceil = \text{An odd integer}$$
$$\Rightarrow \lceil J/2 \rceil = (\text{An odd integer}) - i$$

Case (1) at i = even, $\lceil J/2 \rceil$ = odd,

$$\Rightarrow J = \begin{cases} 2*(\text{An odd integer}), \text{ or} \\ 2*(\text{An odd integer}) - 1 \end{cases} \quad (11)$$

from (10) and (11),

$$\Rightarrow M_{i+1} = \begin{cases} 4*(\text{An odd integer}) - 2K''C - 1 - M_i, \text{ or} \\ 4*(\text{An odd integer}) - 2 - 2K''C - 1 - M_i \end{cases} \quad (12)$$

Case (2) at i = odd, $\lceil J/2 \rceil$ = even,

$$\Rightarrow J = \begin{cases} 2*(\text{An even integer}), \text{ or} \\ 2*(\text{An even integer}) - 1 \end{cases} \quad (13)$$

from (10) and (13),

$$\Rightarrow M_{i+1} = \begin{cases} 4*(\text{An even integer}) - 2K''C - 1 - M_i, \text{ or} \\ 4*(\text{An even integer}) - 2 - 2K''C - 1 - M_i \end{cases} \quad (14)$$

We can obtain Scheme A by setting $K'' = 0$ in (12) and (14):

$$M_{i+1} = \begin{cases} 4*(\text{An odd integer}) - 1 \pm 1 - M_i, & \text{for } i = \text{even} \\ 4*(\text{An even integer}) - 2 \pm 1 - M_i, & \text{for } i = \text{odd} \end{cases}$$

After simplifying,

$$M_i := \begin{cases} 4*(\text{An odd integer}) - 2 \pm 1 - M_{i-1}, & \text{for } i = \text{odd} \\ 4*(\text{An even integer}) - 2 \pm 1 - M_{i-1}, & \text{for } i = \text{even} \end{cases}$$

where

$$1 \le M_i \le 2C, \quad \text{for } i := 0, 1, \ldots, 2R-1$$

To derive Scheme B, let

$$M_i := \begin{cases} 1, \text{ for } i = \text{even} \\ 2C, \text{ for } i = \text{odd} \end{cases}$$

Case (1) at i = odd, from (9) and (13),

$$J = K''C + \frac{1 + 2C + 1}{2} = \begin{cases} 2*(\text{An even integer}), \text{ or} \\ 2*(\text{An even integer}) - 1 \end{cases}$$
$$\Rightarrow J = K''C + C + 1 \quad \in \{3, 4, 7, 8, \ldots\} \quad (15)$$

out of the three possible values of K'', -1, 0 and $+1$, only $K'' = 0$ can satisfy both (15) and $(1 \le J \le 2C)$, therefore,

$$J = C + 1 \in \{3, 4, 7, 8, \ldots\}$$
$$\Rightarrow C \in \{2, 3, 6, 7, \ldots\}$$
$$\Rightarrow C := \begin{cases} 2*(\text{An odd integer}), \text{ or} \\ 2*(\text{An odd integer}) + 1 \end{cases} \quad (16)$$

Case (2) at i = even, from (9) and (11),

$$J = K''C + \frac{2C+1+1}{2} = \begin{cases} 2*(\text{An odd integer}), \text{ or} \\ 2*(\text{An odd integer}) - 1 \end{cases}$$

$$J = K''C + C + 1 \quad \in \{1, 2, 5, 6, \ldots\} \tag{17}$$

both $K'' = 0$ and -1 can satisfy (17) and ($1 \leq J \leq 2C$). When

$$K'' = 0, C \quad \in \{0, 1, 4, 5, \ldots\} \tag{18'}$$

When

$$K'' = -1, C \coloneqq \text{Any positive integer} \tag{18}$$

For all values of i, both (16) and (18) can be satisfied simultaneously by the requirement:

$$C \coloneqq \begin{cases} 2*(\text{An odd integer}), \text{ or} \\ 2*(\text{An odd integer}) + 1 \end{cases}$$
$$\in \{2, 3, 6, 7, \ldots\}$$

which is Requirement (2) of Scheme B.

Now let us consider the diagonal comparison. We will show that Requirement (3) can actually be waived when marking scheme B is used with Algorithm II.

Again, from Fig. 7, items $a_{i,J}$ and $a_{i',J'}$ are compared diagonally if

$$J' = J - (-1)^J \tag{3}$$

$$i' = i - (-1)^{i+\lfloor J/2 \rfloor} \tag{6}$$

Converting J and J' into j and j' using (2) and (3), we obtain

$$j = [4C + M_i - J + (-1)^j t \text{ MOD } 2C] \text{ MOD } 2C$$
$$j' = [4C + M_{i'} - J' + (-1)^{j'} t \text{ MOD } 2C] \text{ MOD } 2C$$
$$= [4C + M_{i'} - J + (-1)^J + (-1)^{j'} t \text{ MOD } 2C] \text{ MOD } 2C$$

The heads and tails of the loops are compared by the diagonal comparisons if

$$j = 2C - 1$$
$$j' = 0$$
$$i' = i + 1$$

Substituting these values into the above expressions, we get

$$j = 2C - 1 \Rightarrow 4C + M_i - J + (-1)^j t \text{ MOD } 2C$$
$$= K*2C + 2C - 1$$
$$j' = 0 \Rightarrow 4C + M_{i+1} - J + (-1)^J + (-1)^{j+1} t \text{ MOD } 2C$$
$$= K'*2C$$

Adding up the two expressions,

$$8C + M_i + M_{i+1} - 2J + (-1)^J = (K + K' + 1)*2C - 1$$
$$\Rightarrow M_i + M_{i+1} = 2J - K''*2C - 1 - (-1)^J$$
$$= 2(J - K'')*C - 1 - (-1)^J$$
$$= 2(J - K'')*C \text{ or } 2(J - K'')*C - 2$$
$$= \text{An even integer}$$

This shows that Requirement (3) can be waived when Scheme B is used with Algorithm II, because if $M_i + M_{i+1}$ equals an odd integer, then the tail-and-head comparisons will be provided by the vertical comparison, but if $M_i + M_{i+1}$ equals an even integer, then it will be provided by the diagonal comparison, as shown above.

The requirements for the marking schemes are summarized in Table 1.

Table 1. Requirements for the Marking Schemes

Marking Scheme	Algorithm I	Algorithm II
A	Requirement 1	Requirement 1
B	Requirements 1 and 2	Requirements 1 and 2
Others	Requirements 1, 3 and others derived from (5), (6) and (9).	Requirement 1 and others derived from (5), (6) and (9)

4.3 Correctness of the termination method

If the marking scheme is correct and Requirements (1), (2) and (3) are duly met, then Condition (1) of Section 3.2 is sufficient to guarantee proper termination. The reason is that, as we can see from expressions (2), (4) and (6), the comparison pattern repeats every $2C$ cycles. If there is no exchange in the most recent consecutive $2C$ cycles, then there will be no further exchange in the subsequent comparisons, meaning that the sorting process must have been completed.

4.4 Timing complexities

Supposing each comparison operation requires time t_C and each shifting operation takes time t_S, then the cycle time of Algorithm I is $(2t_C + t_S)$ and that of Algorithm II is $(3t_C + t_S)$. Let N_1, N_2, T_1 and T_2 be the number of cycles and total sorting times of Algorithms I and II, respectively, then

$$\begin{aligned} T_1 &= (2t_C + t_S)*N_1 \\ T_2 &= (3t_C + t_S)*N_2 \end{aligned} \tag{19}$$

If the input list is already in the desired order, then $N_1 = N_2 = 2C$, which is the number of cycles needed by the control unit to generate the termination signal. For randomly and inversely ordered input lists, both N_1 and N_2 are found to be in $O(N)$ by simulations,[10] and N_2 is observed to be always much less than N_1 (the examples given in Figs 2(a) and 2(b) help illustrate this point).

5. DISCUSSION

Since sorting is such a common and necessary operation in computer applications, there are dozens of sorting algorithms described in the literature. In this paper we have presented two sorting algorithms using the systolic idea, and the functional design of a sorter based on these algorithms has also been suggested. Our primary goal is to look into the design of a special-purpose VLSI chip that can be attached to a conventional host computer such as the one envisioned by Foster and Kung[11] (see Fig. 9).

Undoubtedly, the usefulness of the sorter is not limited to scientific computations, it could also be used in office information systems and relational database machines.

With the stated goal in mind, we now compare our proposal with some existing ones and the following

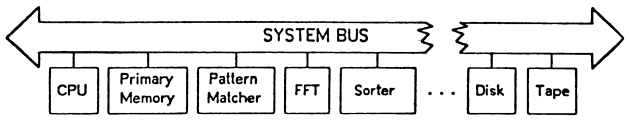

Figure 9. A general-purpose computer with special-purpose chips attached.[11]

criteria will be used: (1) time complexity; (2) hardware complexity and (3) control structure:

(1) *Time complexity*. In Table 2, the sorting times of the various existing algorithms could be divided into four categories: $O(\log N)$, $O(\log^2 N)$, $O(N^{1/2})$ and $O(N)$, where N is the number of items to be sorted. Muller and Preparata's algorithm[3] is in the fastest category, but their algorithm requires a discouraging number of comparators, $O(N^2)$. Batcher's bitonic sorter[1] and the perfect shuffle[5] are in the $O(\log^2 N)$ category, and they are characterized by the shuffle-exchange type of interconnections. The two mesh sorting schemes sort N^2 items on a $N \times N$ mesh with approximately $O(N)$ time, therefore they belong to the $O(N^{1/2})$ category. Nassimi and Sahni's mesh sorting scheme[6] is based on Batcher's bitonic merge algorithm and it needs approximately $14N$ routing steps and $2\log N$ compare–exchange steps on an $N \times N$ mesh, but it requires that the input subfiles be pre-sorted. Thompson and Kung's mesh sorting scheme[9] needs roughly $6N + O(N^{2/3}\log N)$ routing steps and $N + O(N^{2/3}\log N)$ compare–exchange steps. The odd–even sort and the RSS algorithms belong to the $O(N)$ category, but because of their simple control structures and near-neighbour type of data movements, their actual sorting times could be less than those of the mesh sorting schemes which require rather complex control and data movements.

(2) *Hardware complexity*. Sorters with shuffle-exchange type of interconnections are not well-suited to VLSI implementations because shuffle-exchange networks have a very low degree of regularity and modularity, and they require wires of various length. It has been shown by Thompson[12] that at least $O(N^2/\log^2 N)$ chip area is required to lay out an N-vertex shuffle-exchange network. This would be a serious drawback when N is large. On the other hand, the interconnection patterns required by the mesh, the odd–even and the RSS algorithms are highly regular and repetitive, and therefore are conducive to VLSI implementations.

(3) *Control structure*. The logic for the various operations (horizontal comparison, vertical comparison and diagonal comparison) specified in Fig. 5 can be hardwired into each of the RSS comparators, and the control unit shown in Fig. 1 simply broadcasts to all the RSS comparators the sequence of these operations. The control structure required by the RSS algorithms is therefore comparable to that required by the Batcher's and the odd–even sorters, and is very much simpler than that required by the mesh sorters. When implemented as a single chip, the RSS array would need very simple control lines.

Although the RSS array is analogous to the odd–even sorter, its resources are better used because of its recirculating nature. Most existing sorting networks impose non-trivial constraints on their network sizes. For examples, the Batcher's sorter and the perfect shuffle network require that the numbers of input lines be a power of two, and the mesh sorting algorithms operate on square arrays. The constraints of the RSS algorithms (see Table 1) appear to be less stringent in this respect.

In summary, although the RSS algorithms are not optimal in every aspect, they are better than other existing schemes as far as hardware simplicity and hence large-scale implementations are concerned.

Acknowledgement

This research was supported by The National Science and Engineering Research Council of Canada, under grant # 67-9054.

Table 2. Complexities of some sorting algorithms

Method	Input	Number of comparators	Time	Interconnection complexity*	Control complexity*
Batcher's bitonic sorter[1]	N	$O(N\log^2 N)$	$O(\log^2 N)$	high	low
Muller and Preparata's[3]	N	$O(N^2)$	$O(\log N)$	low	high
Odd–even transposition sort[4]	N	$O(N^2)$	$O(N)$	low	low
Perfect shuffle[5]	N	$O(N)$	$O(\log^2 N)$	high	low
Thompson and Kung's mesh sorter[9]	N^2	$N \times N$ mesh	$O(N)$†	low	high
Nassimi and Sahni's mesh sorter[6]	N^2	$N \times N$ mesh	$O(N)$†	low	high
RSS	N	$O(N)$	$O(N)$	low	low

* In terms of amenability to VLSI implementations.
† See discussions.

REFERENCES

1. K. E. Batcher, Sorting networks and their applications. *Proc. AFIPS 1968, Spring Joint Computer Conference,* pp. 307–314, April (1968).
2. H. T. Kung and C. E. Leiserson, Systolic arrays for VLSI. Department of Computer Science, Carnegie-Mellon University, *Technical Report CS-79-103,* April (1979).
3. D. E. Muller and F. P. Preparata, Bounds to complexities of networks for sorting and switching. *J. Ass. Comput. Mach.* **22**, 195–201 (1975).
4. D. E. Knuth, *The Art of Computer Programming, Vol. 3, Sorting and Searching.* Addison-Wesley, Reading, Mass. (1973).
5. H. Stone, Parallel processing with the perfect shuffle. *IEEE Trans. Computers* **C20** (2), 153–161 (1971).
6. D. Nassimi and S. Sahni, Bitonic sort on a mesh-connected parallel computer. *IEEE Trans. Computers* **V10-C28** (1), 2–7 (1979).
7. F. Y. Chin and K. S. Fok, Fast sorting algorithms on uniform ladders. *IEEE Trans. Computers* **O29** (7), 618–631 (1980).
8. C. Tung, T. C. Chen and H. Chang, Bubble ladder for information processing. *IEEE Trans. Magn.* **MAG-11**, 1163–1164 (1975).
9. C. D. Thompson and H. K. Kung, Sorting on a mesh-connected parallel computer. *Communications of the ACM* **20** (4), 263–271 (1977).
10. F. S. Wong and M. R. Ito, A systolic sorter and its simulation results. Department of Electrical Engineering, University of British Columbia, *Technical Report,* October (1982).
11. M. J. Foster and H. T. Kung, Design of special-purpose VLSI chips: examples and opinions. Department of Computer Science, Carnegie-Mellon University, *Technical Report,* September (1979).
12. C. D. Thompson. A complexity theory for VLSI. *Ph.D. thesis,* Carnegie-Mellon University, Department of Computer Science (1979).
13. H. T. Kung, Let's design algorithms for VLSI systems. Department of Computer Science, Carnegie-Mellon University, *Technical Report,* January (1979).

Received January 1983