

# Bitonic Sort on a Mesh-Connected Parallel Computer

DAVID NASSIMI AND SARTAJ SAHNI

**Abstract**—An  $O(n)$  algorithm to sort  $n^2$  elements on an Illiac IV-like  $n \times n$  mesh-connected processor array is presented. This algorithm sorts the  $n^2$  elements into row-major order and is an adaptation of Batcher's bitonic sort. A slight modification of our algorithm yields an  $O(n)$  algorithm to sort  $n^2$  elements into snake-like row-major order. Extensions to the case of a  $j$ -dimensional processor array are discussed.

**Index Terms**—Bitonic sort, complexity, mesh-connected parallel computer, parallel sorting, SIMD machine.

## I. INTRODUCTION

BATCHER'S bitonic sort [2], [6, pp. 232–233, 237] is based upon his algorithm to sort a bitonic sequence into nondecreasing order. A sequence  $X = (x_1, x_2, \dots, x_N)$  is said to be *bitonic* [2], [8] if either 1) there is an index  $i$ ,  $1 \leq i \leq N$ , such that  $x_1 \leq x_2 \leq \dots \leq x_i \geq x_{i+1} \geq \dots \geq x_N$  or 2) the sequence can be shifted cyclically so that condition 1) is satisfied. Batcher's algorithm to sort a bitonic sequence  $X$  is to recursively sort the bitonic sequences  $X_{\text{ODD}} = (x_1, x_3, x_5, \dots)$  and  $X_{\text{EVEN}} = (x_2, x_4, x_6, \dots)$  and then perform the comparison-interchanges  $x_1: x_2, x_3: x_4, x_5: x_6, \dots$ . During the comparison-interchange  $x_i: x_{i+1}$ ,  $x_i$  is replaced by the smaller of  $x_i$  and  $x_{i+1}$  and  $x_{i+1}$  becomes the larger of the two. Any sequence  $Y = (y_1, y_2, \dots, y_N)$  may be sorted by recursively sorting  $(y_1, y_2, \dots, y_{\lfloor N/2 \rfloor})$  into nonincreasing order,  $(y_{\lfloor N/2 \rfloor + 1}, \dots, y_N)$  into nondecreasing order (or vice versa) and then sorting the bitonic sequence  $(y_1, y_2, \dots, y_N)$  into nondecreasing order using Batcher's method.

Bitonic sort has been adapted by Orcutt [7] and Thompson and Kung [9] for an  $n \times n$  mesh-connected parallel computer. The computer consists of  $N = n^2$  identical processors configured in a manner similar to the Illiac IV machine [1]. The assumptions we shall be making on the machine model are as follows.

1) It is an SIMD type [4] machine. The  $N = n \times n$  identical processors may be thought of as positioned according to an  $n \times n$  array  $P(0:n-1, 0:n-1)$ . Each processor  $P(i, j)$  is connected to its neighbor processors  $P(i+1, j)$ ,  $P(i-1, j)$ ,  $P(i, j+1)$ , and  $P(i, j-1)$  if they exist. The end-around connections of the Illiac IV are not assumed here.

2) Each processor has three registers: one routing register  $R_r$  and two storage registers  $R_s$ , and  $R_t$ .

Manuscript received February 1, 1978; revised June 6, 1978 and July 24, 1978. This work was supported in part by the National Science Foundation under NSF Grant MCS 76-21024.

The authors are with the Department of Computer Science, University of Minnesota, Minneapolis, MN 55455.

3) A REGISTER INTERCHANGE instruction with time  $= \tau_r$ . Each selected processor unconditionally interchanges the contents of two of its registers. (The same registers are used for all processors.) In our algorithm, only column-selectability and row-selectability of processors is needed.

4) A ROUTE instruction with time  $= \tau_r$ . All processors route the contents of their  $R_r$  to their immediate neighbor in the same direction. Thus, this instruction simply *shifts* the entire  $R_r$ -array (end-off, zero-filled) unit-distance in one of the four directions up, down, left, or right.

5) A COMPARE-INTERCHANGE instruction with time  $= \tau_c$ . All processors do the (hardware) equivalent of the following statement:

**If** SIGN ( $I, S$ ) \* ( $R_r - R_s$ ) < 0  
**then** interchange ( $R_r, R_s$ )

where  $I$  = processor index, and  $S$  = "pass number" of the algorithm. The function SIGN will be specified later. After a compare-interchange instruction, we shall refer to the element in  $R_s$  as the "accepted" element (to be kept by the processor) and the one in  $R_r$  as the "rejected" element (to be routed back). Note that even though *all* processors carry-out this instruction, only  $N/2$  of the processors would be doing "useful work." The result of the other half is "don't care."

The sorting problem studied in [3], [7], and [9] is that of routing the contents of the  $n \times n$  routing registers to destination processors. Each data item is to be routed to a distinct processor. The processors are assumed indexed in some manner and the routing is such that the  $I$ th processor is to contain the  $I$ th smallest element,  $1 \leq I \leq N$ . Three different indexing schemes have been considered by Thompson and Kung [9]: row-major, shuffled row-major, and snake-like row-major. In row-major order, the index  $I$  of processor  $P(i, j)$  is  $i * n + j$  (i.e., processors are indexed left to right, top to bottom). In shuffled row-major, the index of a processor is obtained by shuffling its row-major index. For example, if the row-major index in binary is  $b_1 b_2 b_3 b_4 b_5 b_6 b_7 b_8$  then its shuffled index is  $b_1 b_5 b_2 b_6 b_3 b_7 b_4 b_8$ . Snake-like row-major indexing is obtained by indexing the processors by row (as in row-major). Processors on even rows are indexed left to right while those on odd rows are indexed right to left (recall that rows are numbered 0 through  $n-1$ ).

Thompson and Kung [9] present fast parallel algorithms for sorting into snake-like row-major and shuffled row-major order. For snake-like row-major order, they present an  $s^2$ -way merge algorithm requiring  $6n + O(n^{2/3} \log n)$  routing steps and  $n + O(n^{2/3} \log n)$  comparison-

