Parallel Sorting Algorithm Using Multiway Merge and Its Implementation on a Multi-Mesh Network

Bhabani P. Sinha[1] and Amar Mukherjee

School of Computer Science, University of Central Florida, Orlando, Florida 32816

Received December 8, 1998; revised November 9, 1999; accepted February 24, 2000

In this paper, we present a parallel sorting algorithm using the technique of multi-way merge. This algorithm, when implemented on a t dimensional mesh having n^t nodes ($t>2$), sorts n^t elements in $O((t^2-3t+2)n)$ time, thus offering a better order of time complexity than the $[((t^2-t)n\log n)/2+O(nt)]$-time algorithm of P. F. Corbett and I. D. Scherson (1992, *IEEE Trans. Parallel Distrib. Systems* **3**, 626–632). Further, the proposed algorithm can also be implemented on a Multi-Mesh network (1999, D. Das, M. De, and B. P. Sinha, *IEEE Trans. Comput.* **48**, 536–551) to sort N elements in $54N^{1/4}+o(N^{1/4})$ steps, which shows an improvement over $58N^{1/4}+o(N^{1/4})$ steps needed by the algorithm in (1997, M. De, D. Das, M. Ghosh, and P. B. Sinha, *IEEE Trans. Comput.* **46**, 1132–1137).

Key Words: multi-way merge; shear-sort; odd–even merge sort; 2D mesh; Multi-Mesh; multidimensional mesh; SIMD; MIMD; PRAM; EREW; CREW.

1. INTRODUCTION

Parallel algorithms for sorting are usually developed on either the network model or the PRAM model. The network model is the more restrictive one. Several interesting results on the complexity of parallel sorting algorithms can be found in [1, 2, 4, 6, 17, 25, 28, 31]. Performance analysis of different parallel sorting algorithms from actual implementation results was reported by Dusseau *et al.* in [13]. Parallel merging on an EREW PRAM model was done by Cole [6] in $O(\log N)$ time for merging a total of N elements using N processors. Akl and Santoro [4] developed a self-reconfiguring optimal algorithm for parallel merge sort on the EREW PRAM model without memory conflicts. This algorithm requires $O((N/P+\log^2 P)\times\log N)$ time for sorting N elements using P processors. Wen proposed a parallel algorithm [34] for multiway merging of k sorted lists ($k\geqslant 2$) using P processors on a CREW PRAM model which requires $O(\log N+\frac{N\log k}{p})$ time, N being the total size of the input lists.

[1] On leave from Indian Statistical Institute, Calcutta, India.

Sorting N elements in $O(N)$ time using a linear array of N processors was proposed in [22] by Mukhopadhyay and Ichikawa and in [21] by Mukherjee. Ajtai *et al.* proposed a sorting network [3] that sorts in $O(\log N)$ time, but the constant in the running time of this algorithm was very large. Batcher's odd-even merge sort [5] can sort N numbers in $O(\log^2 N)$ steps on an N-node hypercube, as well as on bounded-degree derivative networks of the hypercube, commonly known as hypercubic networks [18]. An algorithm for sorting N items on a hypercubic network having P nodes in $O(\frac{\log N \log P}{\log(P/N)})$ steps was proposed by Preparata [26] for an architecture-independent setting and was implemented by Nassimi and Sahni [23]. An algorithm for sorting N packets on a P-node hypercubic network, when $N > P$, has been described in [1, 9]. An $O(\log N \log \log N)$ step algorithm for sorting N elements on hypercubic networks has been proposed by Cypher and Plaxton [8] which is based on merging $\sqrt{N}$ lists of $\sqrt{N}$ items each. Randomized $O(\log N)$ step algorithms for sorting on hypercubic networks have been described in [18, 20, 27].

Many interesting results also exist in the literature for parallel sorting on mesh-like architectures, which are popular due to their structural simplicity. Thompson and Kung [32] developed an $O(\sqrt{N})$ time sorting algorithm on a mesh connected SIMD computer without any wrap-around connections. Schnorr and Shamir [30] proposed a $3\sqrt{N}$ time algorithm on an MIMD mesh model. Scherson and Sen [29] developed an optimal algorithm on an SIMD mesh model, called shear-sort algorithm, that needs $4\sqrt{N} + o(\sqrt{N})$ steps and also a $3\sqrt{N}$ step algorithm on the more powerful MIMD model. Leighton's *column-sort* algorithm [17] uses seven phases to sort N items on an $r \times s$ mesh into column-major order, where $r \geqslant s^2$. Algorithms for optimal sorting on a multidimensional mesh using the MIMD model were proposed by Kunde [15, 16]. Nigam and Sahni [24] have shown that it is possible to sort N^2 numbers on an $N \times N$ mesh using a number of routes that is very close to the distance lower bound for both bidirectional and unidirectional meshes. Leighton has given an algorithm [18] that sorts N elements in $O(N^{1/k})$ steps on a k-dimensional $N^{1/k}$-sided array. Corbett and Scherson [7] extended the idea of the shear-sort algorithm given in [29] for both SIMD and MIMD models to sort $N = n^k$ elements on a k-dimensional mesh (without wrap-around connections) having n^k nodes in $((k^2-k)\, n \log n)/2 + O(nk)$. On the multidimensional mesh architecture, this appears to be so far the best result which would require $6n \log n + O(n)$ steps to sort n^4 elements. The presence of wrap-around connections would not, however, improve the order of the time complexity. The algorithm due to Tsai *et al.* [33] takes $O(\log N + 2(\log^3 N/\log m) - 3\log^2 N + \log N \log m)$ time to sort N elements on a mesh-connected computer with multiple broadcasting (MCCMB) using Nm processors, where m is the number of processors in each dimension of the MCCMB. A sorting algorithm on the Multi-Mesh (MM) network has been proposed by De *et al.* [12] which sorts N data elements on a MM [10, 11] SIMD model in $58N^{1/4} + o(N^{1/4})$ time.

In this paper, we first propose an algorithm for parallel multi-way merge which is implementable on a suitable sorting network. When implemented on a t-dimensional mesh having n^t nodes ($t > 2$), it requires $O((t^2 - 3t + 2)\, n)$ time to sort n^t elements, offering an improved order of time complexity compared to that in [7].

We then show an implementation of this algorithm on a MM network having $N = n^4$ nodes ($n > 2$) to sort N elements in $54N^{1/4} + o(N^{1/4})$ steps, which is an improvement over that of the algorithm in [12] due to a smaller constant factor. This implementation is essentially an amalgamation of two things—the SIMD algorithm in [29] to sort individual $n \times n$ 2D blocks of the MM network and the n-way parallel merging technique, as developed in the first part of this paper, for merging these blocks.

2. PARALLEL MULTI-WAY MERGE SORT

In this section, we first describe an algorithm for parallel n-way merging to merge n given sorted sequences on a network. Next we use this merging algorithm to sort any given set of elements.

2.1. Algorithm for Parallel n-Way Merging

Input. n sorted sequences $S_1, S_2, ..., S_n$, each of length p such that

$$S_1: \{s_{11} \leqslant s_{12} \leqslant s_{13} \leqslant \cdots s_{1p}\}$$
$$S_2: \{s_{21} \leqslant s_{22} \leqslant s_{23} \leqslant \cdots s_{2p}\}$$
$$\cdots$$
$$S_n: \{s_{n1} \leqslant s_{n2} \leqslant s_{n3} \leqslant \cdots s_{np}\}.$$

Output. A sorted sequence V of length np.

PROCEDURE n-MERGE($S_1, S_2, ..., S_n, V$)

Step 1. If $p \leqslant n$, then arrange the elements of the given sequences in the form of an $n \times p$ matrix; sort this matrix using a suitable algorithm (see Section 2.5) to form the sequence V and stop.

Step 2. From the sequence S_i ($\forall i, 1 \leqslant i \leqslant n$), construct n sorted subsequences $S_{i1}, S_{i2}, S_{i3}, ..., S_{in}$, such that $S_{ik} (k \neq n)$ consists of elements $\{s_{il} \mid s_{il} \in S_i$ and $l \bmod n = k\}$ and $S_{in} = \{s_{il} \mid s_{il} \in S_i$ and $l \bmod n = 0\}$. That is,

$$S_{i1} = \{s_{i1} \leqslant s_{i,n+1} \leqslant s_{i,2n+1} \leqslant \cdots\}$$
$$S_{i2} = \{s_{i2} \leqslant s_{i,n+2} \leqslant s_{i,2n+2} \leqslant \cdots\}$$
$$\cdots$$
$$S_{i,n-1} = \{s_{i,n-1} \leqslant s_{i,2n-1} \leqslant \cdots\}$$
$$S_{in} = \{s_{in} \leqslant s_{i,2n} \leqslant \cdots\}.$$

Note. If $p = L * n + j$, where $0 \leqslant j < n$ (L being an integer), then S_{ik} will contain $L+1$ elements if $k \leqslant j$ and L elements if $k > j$; $\forall k, 1 \leqslant k \leqslant n$.

Step 3. For $i=1$ to n do in parallel

$$n\text{-Merge}(S_{1i}, S_{2i}, \ldots, S_{ni}, U_i).$$

Note. The sorted sequence U_k contains $(L+1)n$ elements if $k \leqslant j$ and Ln elements if $k > j$. Thus, if $j=0$, then U_k contains Ln elements $\forall k$; otherwise all U_k's will not have the same length.

Step 4.

Step 4.1. If $j \neq 0$, then increase the length of all the smaller sequences U_k's to the maximum length $(L+1)n$ by inserting (the necessary number of) a very large number, say W. For the sake of notational simplicity, we denote these modified sequences also by U_k, $1 \leqslant k \leqslant n$.

Step 4.2. Arrange the sequences $U_1, U_2, \ldots, U_n$ in the form of an $n \times c$ matrix U, where $c=(L+1)n$ if $j \neq 0$ and Ln otherwise. Thus, for $j \neq 0$,

$$U = \begin{pmatrix} U_1 \\ U_2 \\ \vdots \\ U_n \end{pmatrix} = \begin{pmatrix} u_{11} & u_{12} & u_{13} & \cdots & u_{1,(L+1)n} \\ u_{21} & u_{22} & u_{23} & \cdots & u_{2,(L+1)n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ u_{n1} & u_{n2} & u_{n3} & \cdots & u_{n,(L+1)n} \end{pmatrix}$$
$$= (A_1 \quad A_2 \quad A_3 \quad \cdots \quad A_{L+1}),$$

where each A_i, $1 \leqslant i \leqslant (L+1)$, is a submatrix of size $n \times n$, and for $j=0$, $U=(A_1\, A_2\, A_3 \cdots A_L)$.

Step 5.

Step 5.1. Sort the following pairs of matrices separately in parallel in a column-major order, replacing the original matrices by the sorted ones:

$$[A_1 A_2], [A_3 A_4], [A_5 A_6], \ldots.$$

Step 5.2. Sort the following pairs of matrices separately in parallel in a column-major order, replacing the original matrices by the sorted ones:

$$[A_2 A_3], [A_4 A_5], [A_6 A_7], \ldots.$$

The resulting matrix

$$V = (A_1 \quad A_2 \quad A_3 \quad \cdots),$$

when read in column-major order, gives the required merged output.

Note. We refer to Step 5 (Steps 5.1 and 5.2 taken together) as the *odd-even matrix-merge* operation (this has a sort of conceptual resemblance with the Weavesort in [21]).

2.2. *Correctness of the Algorithm*

We prove the correctness of the procedure *n-Merge* by applying the 0-1 principle [14]. We assume that the given n sorted sequences $S_1, S_2, ..., S_n$ are sequences consisting of only 0's and 1's.

Clearly, for $p \leqslant n$, *n-Merge* terminates successfully. Because of its recursive nature, we assume that merging of subsequences S_{ik}, $1 \leqslant i \leqslant n$, to get U_k, $1 \leqslant k \leqslant n$, is done properly at Step 3.

Let us now assume that the sequence S_i contains l_i number of 0's, $1 \leqslant i \leqslant n$, so that

$$l_i = q_i * n + r_i; \qquad 0 \leqslant r_i \leqslant n-1,$$

where q_i is an integer.

Hence, the subsequences $S_{i1}, S_{i2}, ..., S_{i,r_i}$ will contain $q_i + 1$ number of 0's, while the subsequences $S_{i,r_i+1}, S_{i,r_i+2}, ..., S_{in}$ will contain q_i number of 0's, $\forall i$, $1 \leqslant i \leqslant n$.

Further, since the input elements are taken from the set $\{0, 1\}$, the equivalent action for Step 4.1 is to pad every sequence U_i, $1 \leqslant i \leqslant n$, which will be of length less than $(L+1)n$, with the required number of 1's only.

In view of the above arguments, the number of 0's present in U_i will be less than or equal to that in U_{i-1}, $1 < i \leqslant n-1$. That is, U_1 will contain the maximum number of 0's and U_n will contain the minimum number of 0's.

Let $Q = q_1 + q_2 + \cdots + q_n$. Hence, we have the following observations:

(1) All the sequences $U_1, U_2, ..., U_n$ will contain at least Q number of 0's.

(2) The sequence U_1 will contain the maximum number of 0's among all these sequences and that number is at most $Q+n$.

(3) The sequence U_n will contain the minimum number of 0's and that number is at least Q.

According to our assumption, merging will be done successfully in Step 3. As a result, each of $U_1, U_2, ..., U_n$ is a sorted sequence of 0's followed by 1's.

Hence, we get that the first Q columns of the matrix U consist of 0's only. The next n columns (that is, starting from column number $Q+1$ to column number $Q+n$) may contain both 0's and 1's and the remaining columns contain 1's only. The columns containing either only 0's or only 1's will be referred to as *clean columns* and those containing both 0's and 1's will be called *dirty columns*. That is, the matrix U contains at most n dirty columns and all these dirty columns appear consecutively.

We are now required to prove that the matrix U can be sorted successfully in Steps 5.1 and 5.2.

Let D be the submatrix of U containing only dirty columns. D can contain at most n columns; i.e., the dimension of D can be at most $n \times n$. If we can sort D properly, the matrix U will also be sorted. We now consider the following two cases:

Case 1. D is contained completely within one of the submatrices $[A_1A_2]$, $[A_3A_4]$, $[A_5A_6]$, ... Then after Step 5.1, the matrix U will be sorted.

Case 2. D is contained partly in $[A_{2i-1}A_{2i}]$ and partly in $[A_{2i+1}A_{2i+2}]$. That is, D is actually contained partly in both A_{2i} and A_{2i+1}. After Step 5.1, each of the submatrices $[A_{2i-1}A_{2i}]$ and $[A_{2i+1}A_{2i+2}]$ will contain at most one dirty column and these dirty columns will lie within the last n columns of the submatrix $[A_{2i-1}A_{2i}]$ and within the first n columns of $[A_{2i+1}A_{2i+2}]$. That is, after Step 5.1, the two dirty columns are contained in $[A_{2i}A_{2i+1}]$. These dirty columns will be cleaned after the execution of Step 5.2. Hence the proof.

2.3. *Implementation of n-Way Merge on a Network*

A network implementation of the procedure n-Merge is shown in Fig. 1, for $n=3$ and $p=9$. The *odd-even matrix-merge* modules can be designed in any suitable manner, e.g., using a 3-dimensional $n \times n \times 2$ mesh network or a hypercubic network. The overall time complexity and also the number of comparators required for merging would depend on the actual implementation of this module. In a latter

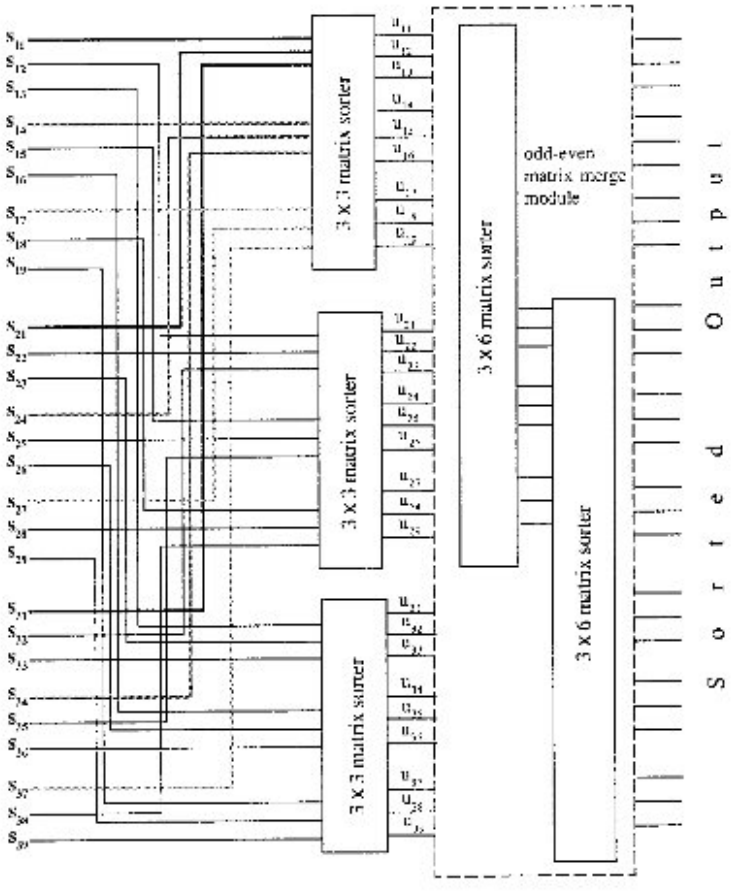

FIGURE 1

section, we compute the time complexity in terms of the time needed by this module and also an $n \times n$ matrix sorter.

2.4. *Sorting by n-Way Merging*

Input. A set $X = \{x_1, x_2, ..., x_N\}$ of N elements. (We assume that $N = n^t$ for some $t \geqslant 1$; otherwise we append to X the required number of elements all of which are greater than any element present in X).

Output. A sorted sequence.

PROCEDURE MULTIWAY-MERGE-SORT

Step 1. If $N = n$, then sort the elements and stop; otherwise if $N = n^2$, then arrange the elements in the form an $n \times n$ matrix, sort this matrix by a suitable algorithm (see Section 2.5), and then stop.

Step 2. Divide the set X into n subsets of equal length N/n.

Step 3. Sort each of the subsets in parallel by a recursive call to procedure *Multiway-Merge-Sort*.

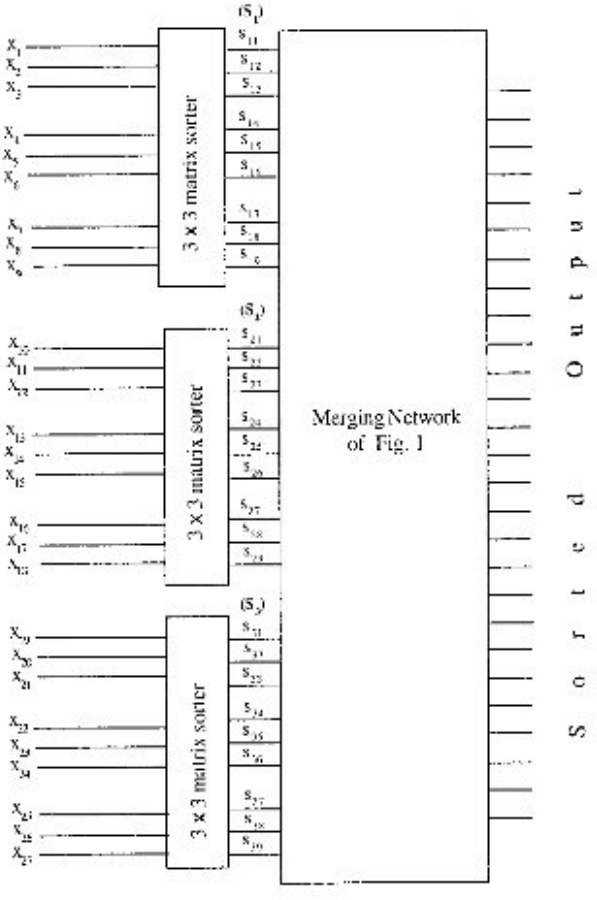

FIGURE 2

Step 4. Merge n sorted sequences obtained in Step 3 applying the procedure n-Merge to get the final sorted sequence.

The correctness of the procedure *Multiway-Merge-Sort* is quite apparent. An example of its network implementation is shown in Fig. 2 for $n=3$ and $N=27$.

2.5. *Timing Analysis*

Let $p=n^r$ and $M(p\times n)$ denote the time taken to merge n sorted sequences, each of length p by the procedure n-Merge. Moreover, let us assume that $S(n\times n)$ and $S(n\times 2n)$ denote the time taken to sort a matrix of size $n\times n$ and a matrix of size $n\times 2n$, respectively. We then have the following relations from the steps of the procedure n-Merge:

$$\begin{aligned} M(p\times n) &= M\left(\frac{p}{n}\times n\right)+2S(n\times 2n)\\ &= M\left(\frac{p}{n^2}\times n\right)+4S(n\times 2n)\\ &= \cdots\cdots\\ &= M(n^2\times n)+2(r-2)\,S(n\times 2n)\\ &= S(n\times n)+2(r-1)\,S(n\times 2n). \end{aligned}$$

Let $MMS(N)$ denote the time taken to sort a sequence of length N by the procedure *Multiway-Merge-Sort*. Hence, we can write

$$\begin{aligned} MMS(n^t) &= MMS(n^{t-1})+M(n^{t-1}\times n)\\ &= MMS(n^{t-1})+S(n\times n)+2(t-2)\,S(n\times 2n)\\ &= MMS(n^{t-2})+2S(n\times n)+\{2(t-2)+2(t-3)\}\,S(n\times 2n)\\ &= \cdots\cdots\\ &= MMS(n^2)+(t-2)\,S(n\times n)+2\{(t-2)+(t-3)+\cdots+1\}\,S(n\times 2n)\\ &= (t-1)\,S(n\times n)+(t^2-3t+2)\,S(n\times 2n). \end{aligned}$$

If a hypercubic network is used to sort the $n\times n$ and $n\times 2n$ matrices in steps 1 and 4 of the procedure *Multiway-Merge-Sort*, then $MMS(n^t)$ will be equal to $O(t^2\log^2 n)$. On the other hand, a t-dimensional mesh with n^t nodes can be used to store the data elements, and then the $n\times n$ and $n\times 2n$ matrices can be sorted in $O(n)$ time using the algorithm in [29]. As an example, consider the case of n^3 elements in a 3D $n\times n\times n$ mesh, with nodes organized along x, y, and z directions, respectively. Initially, the n^2 data elements in every plane parallel to the xy plane will be sorted in x-major order in $O(n)$ time. This will form the sets S_i's, $i=1, 2, \ldots, n$, each set being constituted by the elements residing in a plane parallel to the xy plane. At this point, the set of all elements appearing in a plane parallel

to the yz plane will be only those which can form the set U_i for some value of i, $1 \leqslant i \leqslant n$. So, the elements in each plane parallel to the yz plane are now sorted to form the sets U_i's, $i = 1, 2, ..., n$, each U_i being stored in a separate plane in y-major order, say. It then follows that each of the matrices $A_1, A_2, ...,$ will be formed with the elements residing in a plane parallel to the xy plane. Hence, the odd-even matrix-merge operation can be very well performed in $O(n)$ time after this step.

A generalization of the above idea can be made so that if we were initially given n sorted lists, each residing in an $(i-1)$-dimensional hyperplane of an i-dimensional mesh having $n \times n \times \cdots \times n = n^i$ nodes $(i \geqslant 3)$, then the sets U_i's, $i = 1, 2, ..., n$, and also the matrices $A_1, A_2, ..., A_i$, can easily be formed with the elements of the i-dimensional mesh.

It thus follows from the above discussions that the procedure *Multiway-Merge-Sort* can be implemented on a t-dimensional mesh having n^t nodes $(t > 2)$, to sort n^t elements in $O((t^2 - 3t + 2)\, n)$ compare-exchange steps, and thus providing a better order of time complexity than the $[(((t^2 - t)\, n \log n)/2) + O(nt)]$-step algorithm of Corbett and Scherson [7].

3. IMPLEMENTATION OF MULTIWAY-MERGE-SORT ON A MULTI-MESH

In this section, we would show how the Multiway-Merge-Sort algorithm can also be efficiently implemented on a Multi-Mesh [10, 11]. The MM network consists of n^2 meshes of size $n \times n$ each, which themselves are also arranged in the form of an $n \times n$ matrix. Each of these constituent meshes is called a *block*, which is uniquely identified by two coordinates, say, α and β, as $B(\alpha, \beta)$. There are n^4 nodes (processors) in the network each of which is designated using a four-tuple of coordinates. Thus, let $P(\alpha, \beta, x, y)$ denote a processor at the xth row and yth column of the block $B(\alpha, \beta)$. Each of α, β, x, and y can assume values in the range 1 to n (both inclusive).

The processor $P(\alpha, \beta, x, y)$ is connected to its four neighbors $P(\alpha, \beta, x \pm 1, y \pm 1)$, if they exist, using the *intrablock links*. Additional connections called *interblock links* between the boundary and corner processors of different blocks are also used as described in [10]. An example of a Multi-Mesh network for $n = 3$ is shown in Fig. 3, where all the interblock links are not shown.

We assume that there are $N = n^4$ data elements on the n^4 nodes (processors) of the Multi-Mesh. Sorting will be done by applying the n-way merge on these data elements in different stages. We first give the following notations and definitions.

We refer to the directions corresponding to the coordinate values y, x, β, and α as row, column, third, and fourth dimensions, respectively. Let $D(\alpha, \beta, x, y)$ denote the data element residing in the processor $P(\alpha, \beta, x, y)$. Based on the dimension of sorting, we define the following:

DEFINITION 1. By an R-operation, we mean independent row sorts for all the blocks in parallel, where the direction of row sorts of two consecutive rows are opposite within a block and also the same row of two consecutive blocks along the third or fourth dimensions are sorted in opposite directions. If, however, all rows

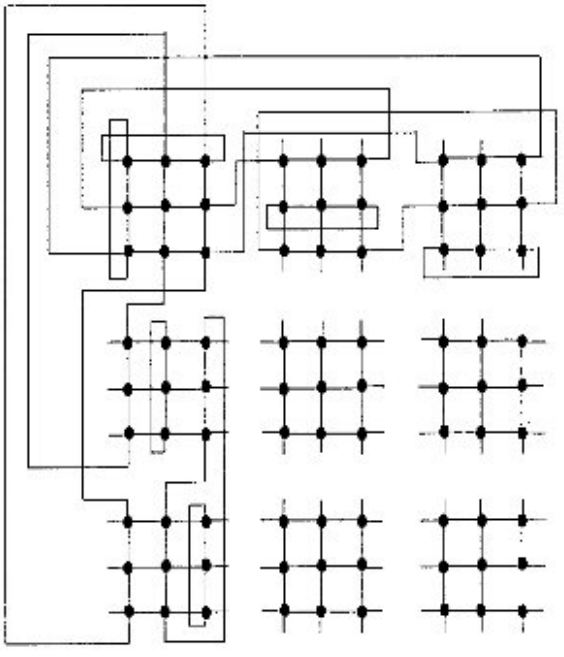

FIGURE 3

in a block are sorted in the same direction, but rows of consecutive blocks are sorted in opposite directions, then we call this an R^*-operation.

DEFINITION 2. By a C-operation, we mean independent column sorts for all the blocks in parallel, where the direction of column sort for all the columns within a block will be the same, but it will be reversed in the next consecutive block along the third or fourth dimensions.

DEFINITION 3. A T-operation sorts each set of n data elements $D(\alpha, *, x, y)$ in parallel over the third dimension ("*" implies all possible values (from 1 to n) of the respective coordinate) so that

$$D(\alpha, 1, x, y) \leqslant D(\alpha, 2, x, y) \leqslant \cdots \leqslant D(\alpha, n, x, y).$$

DEFINITION 4. An F-operation sorts each set of n data elements in parallel over the fourth dimension, so that

$$D(1, \beta, x, y) \leqslant D(2, \beta, x, y) \leqslant \cdots \leqslant D(n, \beta, x, y), \qquad \text{if } \beta \text{ is odd, and}$$

$$D(1, \beta, x, y) \geqslant D(2, \beta, x, y) \geqslant \cdots \geqslant D(n, \beta, x, y), \qquad \text{if } \beta \text{ is even.}$$

DEFINITION 5. By an *even block* we mean an ordered block $B(\alpha, \beta)$ such that $\alpha + \beta$ is an even number. Such a block is sorted in snake-like row-major order [12] in which the sorted sequence starts from the leftmost end of the first row and ends at the nth row.

DEFINITION 6. By an *odd block* we mean an ordered block $B(\alpha, \beta)$ such that $\alpha+\beta$ is an odd number. Such a block is sorted in snake-like row-major order in which the sorted sequence starts from the nth row and ends at the leftmost end of the first row.

DEFINITION 7. A 3D block is a sequence of alternate odd and even blocks for a given value of α such that all the elements of the sorted block $B(\alpha, \beta)$ are less than or equal to all the elements of the sorted block $B(\alpha, \beta+1)$, $\forall\beta$, $1 \leqslant \beta < n$.

We note that there is no direct link among the respective nodes in the MM network to effect the T-operation. Hence, the T-operation is performed in three stages. In the first stage, all data elements are given n horizontal circular shifts (along the horizontal cycles of length n and $2n$ in the MM network [10, 11]), so that the ith rows of the blocks $B(\alpha, *)$ are all brought to the same block $B(\alpha, i)$. The second stage of the T-operation consists of sorting the elements in each of these columns individually. The third stage of the T-operation is just the reverse of the first stage, in which the sorted elements in the jth row of the block $B(\alpha, i)$ are transferred back to the ith row of the block $B(\alpha, j)$ through n horizontal shifts. In general, the first and third stages of the T-operation require a total of $n+n=2n$ routing steps. The second stage can be completed in n steps of odd-even transposition sort.

As in the T-operation, the F-operation also requires three stages. The first stage involves n shifts of all data elements along the vertical cycles present in the MM network so that elements to be sorted appear in a single row of an appropriate block, the second stage consists of sorting each row of the blocks individually, and the third stage would be another n vertical shifts of the data elements.

Since the correctness of the proposed implementation will be proved using the 0-1 principle [14], we assume that all input data elements are taken from the set $\{0, 1\}$. Following the notation in [12], a row of a block will be denoted by ϕ, ω, and δ if it consists of only 0's, only 1's, and a mixture of 0's and 1's, respectively. With these notations, if we consider two consecutive sorted blocks (one is an odd block and the other is an even one) for a given value of α, then these blocks will look like that in Fig. 4a. Hence, by similar arguments as given in [12], we can get the following result.

LEMMA 1. *If there are only n matrices, $A_1, A_2, \ldots, A_n$, generated in Step 4 of the n-Merge algorithm and they are stored in blocks $B(\alpha, 1), B(\alpha, 2), \ldots, B(\alpha, n)$, respectively, for some value of α, then the odd-even matrix-merge operation required for the n-Merge algorithm can be done by first sorting each individual block to generate consecutive odd and even blocks and then applying the $T-C-R-C-R$ operations. Each block will be sorted in snake-like row-major order after this. The operations T and the latter C would each involve only two compare-exchange steps; i.e., the total number of steps required for the $T-C-R-C-R$ operations is $(2n+2)+n+n+2+n=5n+4$.*

Remarks. To get each block sorted in row-major order, we need to change the $T-C-R-C-R$ operations in Lemma 1 to $T-C-R-C-R^*$.

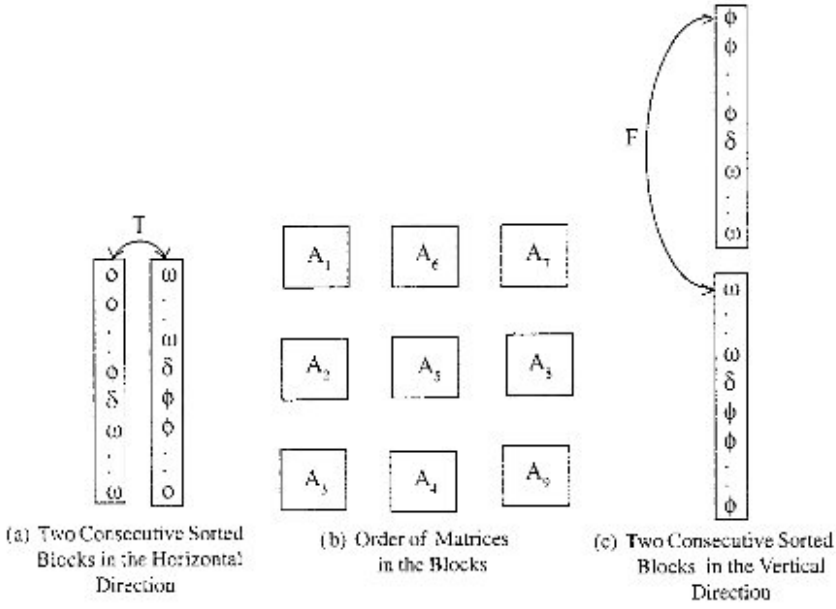

FIGURE 4

On the other hand, if there are n^2 matrices $A_1, A_2, ..., A_{n^2}$ (arising during the process of merging n sorted sequences each of length n^3), then we can store them in the blocks of the Multi-Mesh in the manner shown in Fig. 4b for $n=3$. If the individual blocks are now sorted to generate consecutive odd and even blocks for a given value of α as well as for a given value of β, then all but two blocks will contain only ϕ's or only ω's. The two other blocks may contain at most two δ's, and let us call them *dirty* blocks. If these two dirty blocks appear in the same column of blocks (having the same value of the β coordinate), then by a similar argument as for Lemma 1, we can sort these two blocks applying a sequence of $F-C-R-C-R$ operations (Fig. 4c). However, these two consecutive dirty blocks may also appear in two consecutive columns of blocks. In that case, the sequence of operations $T-C-R-C-R$ would have cleaned the two blocks. Both these cases can be satisfactorily taken into account by a combined $F-T-C-R-C-R$ operation. Note that the elements of the blocks may be exchanged due to either the F-operation or the T-operation, but not both. This is due to the fact that there are only two dirty blocks, and considering that the blocks form a snake-like chain as in Fig. 4b, on one side of these dirty blocks along the chain, there are blocks containing ϕ's only and on the other side there are blocks containing ω's only. We now have the following result.

LEMMA 2. *If there are n^2 matrices $A_1, A_2, ..., A_{n^2}$ generated during the execution of the n-Merge algorithm and these matrices are stored in the blocks of the Multi-Mesh in the order shown in Fig. 4b, then the odd-even matrix-merge operation for these matrices can be completed by first sorting the individual blocks and then applying a sequence of $F-T-C-R-C-R$ operations. The operations F, T, and the latter C would each involve only two compare-exchange steps, giving rise to a total of $7n+6$ compare-exchange/routing steps for the combined $F-T-C-R-C-R$ operations.*

We now describe the algorithm for sorting, on Multi-Mesh in the following steps:

ALGORITHM MM-SORT

/* Construct the initial n^2 sorted lists each containing n^2 elements */

Step 1. Sort each of the n^2 blocks in snake-like column-major order using the shear-sort algorithm in [28]. Rearrange the elements in the columns so that each block is sorted now in column-major order (refer to Fig. 5a for an example with $n=3$).

/* Merging of elements in a row of processor blocks, i.e., in $B(\alpha, *)$, $1 \leqslant \alpha \leqslant n$, starts here */

Step 2. Data elements are shifted horizontally through n places. (The situation after this step is explained by Fig. 5b for $n=3$).

Step 3. Apply the shear-sort algorithm to sort each block in snake-like row-major order, such that for a given α, the sorted sequence in a block $B(\alpha, \beta)$ starts from the leftmost end of the first row for odd values of β and from the rightmost end of the first row for even values of β.

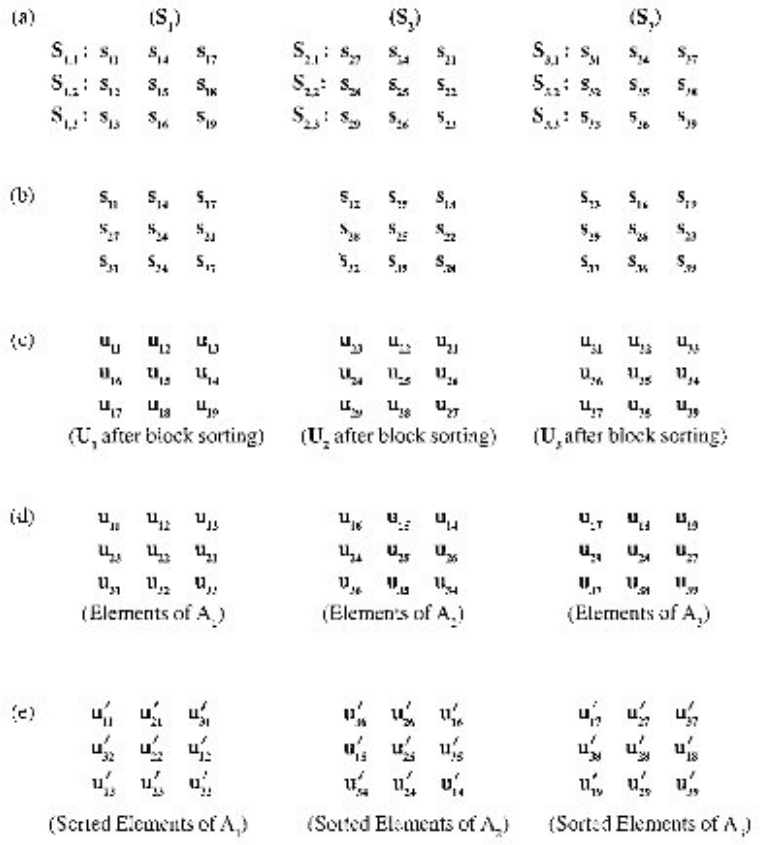

FIGURE 5

Note. After Step 3, we get the sorted sequences $U_1, U_2, ..., U_n$ for merging the n^2 elements in a row of processor blocks (i.e., for a given value of α). Each of these sequences now appears in a single block. An example for $n = 3$ is shown in Fig. 5c.

Step 4. All data elements are again shifted horizontally through n steps. (This will bring all elements of a matrix A_i, $1 \leqslant i \leqslant n$, into a single block of the MM network, as shown in Fig. 5d).

/* odd-even matrix merge for the matrices A_i's in a row of processor blocks starts here */

Step 5. Sort individual blocks again by using the shear-sort algorithm so that each ordered block is now an *odd block* or an *even block*, as shown by the example of Fig. 5e.

Step 6. Apply the $T-C-R-C-R^*$ operations on the data elements using only two compare-exchange steps during the second stage of T-operation. (Fig. 6 shows the sequences V_1, V_2, and V_3 for $n = 3$ after this step).

/* merging of 3D blocks starts here with the given sorted sequences $V_1, V_2, ..., V_n$ */

Step 7. Apply n vertical shifts to all data elements. (After this step, the blocks B(α, $*$) will contain all the elements of the subsequences $V_{i,\alpha}$, $1 \leqslant i \leqslant n$.

Step 8. Apply Steps 1 to 6 above, with the only change in Step 6 as $T-C-R-C-R$, on the data elements to merge $V_{i,\alpha}$'s, $\forall i$, $1 \leqslant i \leqslant n$, for a given α to obtain the new sorted sequences $W'_1, W'_2, ..., W'_n$ in the 3D blocks. (This step completes Step 3 of the n-Merge algorithm.

Step 9. Shift the data elements in individual blocks using only the *intrablock links*, as if the elements in blocks with odd values of α are rotated in the counter-clockwise directions and those in blocks with even values of α are rotated in the clockwise direction with the following structures: original ith row of every block now becomes the ith column for odd α and $(n-i+1)$th column for even α, $1 \leqslant i \leqslant n$.

Step 10. Apply n vertical shifts to all data elements along the vertical cycles in MM. (The individual blocks now store the matrices A_i's required for the final step of merging).

FIGURE 6

1	2	3	54	53	52	55	56	57
6	5	4	49	50	51	60	59	58
7	8	9	48	47	46	61	62	63
18	17	16	37	38	39	72	71	70
13	14	15	42	41	40	67	68	69
12	11	10	43	44	45	66	65	64
19	20	21	36	35	34	73	74	75
24	23	22	31	32	33	78	77	76
25	26	27	30	29	28	79	80	81

FIGURE 7

/* Odd-even matrix-merge for the final step starts here */

Step 11. Sort the individual blocks by using shear-sort algorithm.

Step 12. Apply $F-T-C-R-C-R$ operations with only two compare-exchange steps in each of the F, T, and the last C-operations to sort all the elements in the final order as illustrated in Fig. 7.

The correctness of the algorithm *MM-Sort* follows directly from the correctness of the algorithm n-Merge, that of the algorithm *Multi-Merge-Sort*, and Lemmas 1 and 2.

3.1. *Timing Analysis for MM-Sort*

Step 1 of the algorithm *MM-Sort* requires $(4n+o(n))+n)=5n+o(n)$ steps. Each of the Steps 2, 4, 7, and 10 requires n steps. Sorting in each of the Steps 3, 5, and 11 requires $4n+o(n)$ steps. Step 6 requires $5n+4$ steps. Step 9 requires $n-1$ steps. Step 12 requires $7n+6$ steps.

Hence, Steps 1 through 6 requires a total of $20n+o(n)$ steps. The overall time required for the algorithm *MM-Sort* is $20n+n+20n+(n-1)+n+4n+(7n+3)+o(n)=54n+o(n)$ steps. We now have the following result.

THEOREM 1. *$N=n^4$ elements can be sorted on the Multi-Mesh network using n-way merging, in a total of $54N^{1/4}+o(n)$ compare-exchange/routing steps.*

4. CONCLUSIONS

We have described an algorithm for sorting by using parallel multi-way merge. When implemented on a t-dimensional mesh having n^t nodes, this algorithm takes $O((t^2-3t+2)n)$ time to sort n^t elements, and thus offers a better order of time complexity than the $[(((t^2-t)n\log n)/2)+O(nt)]$-time algorithm of Corbett and Scherson [7]. Then we have given an implementation of the algorithm on a Multi-Mesh network to sort N elements in $54N^{1/4}+o(N^{1/4})$ compare-exchange/routing steps, which is an improvement over the previously known result [12] of $58N^{1/4}+o(N^{1/4})$ steps on the Multi-Mesh.

ACKNOWLEDGMENTS

This work was supported by a grant received under the *Academic Excellence Plan* of the School of Computer Science of the University of Central Florida, Orlando, Florida. The authors thank Robert Franceschini for a careful reading of the manuscript and many comments and suggestions to improve the quality of the paper.

REFERENCES

1. A. Aggarwal and M.-D. Huang, Network complexity of sorting and graph problems and simulating CRCW PRAMs by interconnection networks, *in* "Lecture Notes in Computer Science," Vol. 319, pp. 339–350, Springer-Verlag, Berlin/New York, 1988.
2. M. Aigner, Parallel complexity of sorting problems, *J. Algorithms* **3** (March 1982), 79–88.
3. M. Ajtai, J. Komlos, and E. Szemeredi, An $O(n \log n)$ sorting network, *Combinatorica* **3** (1983), 1–19; also in "Proc. 15th ACM STOC," pp. 1–9, 1983.
4. S. G. Akl and N. Santoro, Optimal parallel merging and sorting without memory conflicts, *IEEE Trans. Comput.* **36**, 11 (November 1987), 1367–1369.
5. K. Batcher, Sorting networks and their applications, *AFIPS Spring Joint Computing Conf.* **32** (1968), 307–314.
6. R. Cole, Parallel merge sort, *in* "Proc. 27th IEEE Symp. FOCS," pp. 511–516, 1986.
7. P. F. Corbett and I. D. Scherson, Sorting in mesh connected multiprocessors, *IEEE Trans. Parallel Distrib. Systems* **3**, 5 (Sept. 1992), 626–632.
8. R. Cypher and G. Plaxton, Deterministic sorting in nearly logarithmic time on the hypercube and related computers, *in* "Proc. 22nd Ann. ACM Symp. Theory of Computing," pp. 193–203, 1990.
9. R. Cypher and J. L. C. Sanz, Optimal sorting in reduced architectures, *in* "Proc. Int. Conf. Parallel Processing, Pennsylvania State Univ., August 1998," Vol. 3, pp. 308–311, 1988.
10. D. Das, M. De, and B. P. Sinha, A new network topology with multiple meshes, *IEEE Trans. Comput.* **48** (May 1999), 536–551.
11. D. Das and B. P. Sinha, Multi-Mesh—An efficient topology for parallel processing, *in* "Proc. Ninth Int. Parallel Processing Symposium, Santa Barbara, April 25–28, 1995," pp. 17–21, 1995.
12. M. De, D. Das, M. Ghosh, and B. P. Sinha, An efficient sorting algorithm on the Multi-Mesh network, *IEEE Trans. Comput.* **46**, 10 (October 1997), 1132–1137.
13. A. C. Dusseau, D. E. Culler, K. E. Schauser, and R. P. Martin, Fast parallel sorting under log P: Experience with the CM-5, *IEEE Trans. Parallel Distrib. Comput.* **7**, 8 (Aug. 1996), 791–805.
14. D. Knuth, "The Art of Computer Programming, Sorting and Searching," Vol. 3, Addison–Wesley, Reading, MA, 1973.
15. M. Kunde, Optimal sorting on multidimensionally mesh-connected arrays, *in* "Proc. Fourth Ann. STACS'87," Lecture Notes in Computer Science, Vol. 247, pp. 408–419, Springer-Verlag, Berlin/New York, 1987.
16. M. Kunde, Routing and sorting in mesh-connected arrays, *in* "Proc. Third Aegean Workshop Computing," Lecture Notes in Computer Science, Vol. 319, pp. 423–433, Springer-Verlag, Berlin/New York, 1988.
17. F. T. Leighton, Tight bounds on the complexity of parallel sorting, *IEEE Trans. Comput.* **34**, 4 (April 1985), 344–354.
18. F. T. Leighton, "Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes," Morgan Kaufmann, San Mateo, CA, 1992.
19. T. Leighton, B. Maggs, A. Ranade, and S. Rao, Randomized routing and sorting in fixed-connection networks, *J. Algorithms* **17**, 1 (1994), 157–205.

20. T. Leighton and G. Plaxton, A (fairly) simple circuit that (usually) sorts, *in* "Proc. 31st Ann. Symp. FOCS," pp. 264–274, Oct. 1990.
21. A. Mukherjee, "Weavesort—A New Sorting Algorithm for VLSI," Tech. Report TR-53, 81, UCF, 1981; also in "Introduction to n-MOS and CMOS VLSI Systems Design," Prentice–Hall, New York, 1986.
22. A. Mukhopadhyay and T. Ichikawa, "An n-Step Parallel Sorting Machine," Tech. Report 72-03, Dept. of Computer Science, Univ. of Iowa, 1972.
23. D. Nassimi and S. Sahni, Parallel permutation and sorting algorithms and a new generalized connection network, *J. Assoc. Comput. Mach.* **29**, 3 (1982), 642–667.
24. M. Nigam and S. Sahni, Sorting n^2 numbers on $n \times n$ meshes, *IEEE Trans. Parallel Distrib. Comput.* **6**, 12 (Dec. 1995), 1221–1225.
25. G. Plaxton, On the network complexity of selection, *in* "Proc. 30th Ann. Symp. FOCS," pp. 396–401, Nov. 1989.
26. F. Preparata, New parallel sorting schemes, *IEEE Trans. Comput.* **27**, 7 (July 1978), 669–673.
27. J. Reif and L. Valiant, A logarithmic time sort for linear size networks, *J. Assoc. Comput. Mach.* **34**, 1 (1987), 60–76.
28. R. Reischuk, A fast probabilistic parallel sorting algorithm, *in* "Proc. 22nd IEEE Symp. FOCS," pp. 212–219, 1981.
29. I. D. Scherson and S. Sen, Parallel sorting in two-dimensional VLSI models of computation, *IEEE Trans. Comput.* **38**, 2 (Feb. 1989), 238–249.
30. C. P. Schnorr and A. Shamir, An optimal sorting algorithm for mesh connected computers, *in* "Proc. 18th ACM STOC," pp. 255–263, May 1986.
31. C. D. Thompson, The VLSI complexity of sorting, *IEEE Trans. Comput.* **32** (Dec. 1983).
32. C. D. Thompson and H. T. Kung, Sorting on a mesh-connected parallel computer, *Comm. Assoc. Comput. Mach.* **20** (Apr. 1977), 263–271.
33. S. S. Tsai, S. J. Horng, S. S. Lee, H. R. Tsai, and T.-W. Kao, An efficient sorting algorithm on mesh connected computers with multiple broadcasting, *in* "Proc. Int. Conf. High Performance Computing," pp. 437–442, Dec. 27–30, 1995.
34. Z. Wen, Multiway merging in parallel, *IEEE Trans. Parallel Distrib. Comput.* **7**, 1 (January 1996), 11–17.

Printed in Belgium