Brief Contributions

An Efficient Sorting Algorithm on the Multi-Mesh Network

Mallika De, Debasish Das, Mabhin Ghosh,
and Bhabani P. Sinha, *Senior Member*, *IEEE*

Abstract—The shear-sort algorithm [19] on an SIMD mesh model requires $4\sqrt{N} + o(\sqrt{N})$ time for sorting N elements arranged on a $\sqrt{N} \times \sqrt{N}$ mesh. In this paper, we present an algorithm for sorting N elements in time $O(N^{1/4})$ on an SIMD Multi-Mesh architecture, thereby significantly improving the order of the time complexity. The Multi-Mesh architecture [23], [24] is built around n^2 blocks, where each block is an $n \times n$ mesh with $n = N^{1/4}$, so that each processor will uniformly have four neighbors in the final topology.

Index Terms—2D mesh, Multidimensional mesh, wrap-around connection, SIMD, MIMD, sorting, shear-sort.

1 Introduction

Several interesting results on the complexity of parallel sorting are given in the literature [1], [2], [3], [4], [5]. Odd-even Merge-Sort can sort N numbers in $O(log^2 N)$ steps on an N-node hypercube [6], as well as on bounded-degree derivative networks of the hypercube, commonly known as hypercubic networks [7], such as butterfly, shuffle-exchange graph, de Bruijn graph, Benes network, and cube-connected cycles (CCC). The $O\left(\frac{logNlogM}{log(N/M)}\right)$ step algorithm for sorting M items on an N-node hypercubic network is described in an architecture-independent setting by Preparata in [8] and was implemented by Nassimi and Sahni in [9]. Algorithms for sorting M packets on an N-node hypercubic network when $M > N$ are described in [10], [11], and [12]. The $O(log\ N\ log\ log\ N)$ step algorithm for sorting on hypercubic networks [13] is based on merging $\sqrt{N}$ lists of $\sqrt{N}$ items. Randomized $O(log\ \ N)$-step algorithm for sorting on hypercubic networks are described in [14], [15], and [16].

Thompson and Kung [17] developed an $O(\sqrt{N})$ time sorting algorithm using a mesh connected SIMD parallel computer without wrap-around connections. Schnorr and Shamir gave a $3\sqrt{N}$ time algorithm on an MIMD mesh model [18]. Finally, Scherson and Sen came up with an optimal $4\sqrt{N} + o(\sqrt{N})$ step algorithm on an SIMD mesh model and a $3\sqrt{N}$ step algorithm on the more powerful MIMD model [19]. Leighton's *Column-sort* algorithm [7] is a seven-phase algorithm that sorts N items in an $r \times s$ mesh into column-major order, where $r \geq s^2$. Algorithms for optimal sorting on a multidimensional mesh using the MIMD model were proposed by Kunde [20], [21]. In a 4D mesh, the required steps on his model will be $O(N^{1/4})$ for sorting N elements. In [7], Leighton has given an algorithm that sorts N elements in $O(N^{1/k})$ steps on a k-dimensional $N^{1/k}$-sided array. The shear -sort algorithm described in [19] was extended by Corbett and Scherson [22] (for both SIMD and MIMD models) to sort $N = n^k$ elements, on a k-dimensional mesh having n^k nodes. The sorting time was $\frac{(k^2-k)nlogn}{2} + O(nk)$. This appears to be, so far, the best known algorithm for sorting on a k-dimensional mesh, using the basic principle of shear-sort from [19]. Thus, in a 4D mesh without wrap-around connections (using SIMD model), this algorithm in [19] would require approximately $6n\ log\ n + o(n)$ steps to sort n^4 elements. Presence of wrap-around connections along all four dimensions would not, however, improve the order of the time complexity.

In this paper, we propose an algorithm for sorting $N = n^4$ elements using the SIMD model of [19] by a Multi-Mesh network [23], [24]. The Multi-Mesh network is different from the multidimensional mesh proposed in [22]. A Multi-Mesh (MM) network having n^4 processors is built around n^2 meshes (blocks) of size $n \times n$ each. The degree of each processor is four for $n > 2$, which is same as that in an Illiac IV architecture, whereas the degree of each processor in a 4D mesh with n^4 nodes is eight (having wrap-around links). The time complexity of the first version of our proposed algorithm is $O(N^{1/4}\ log\ N)$ as opposed to $O(N^{1/2})$ on a 2D mesh [19]. For $N = 4{,}096$, our algorithm requires around 1,070 compare-exchange/routing steps, whereas the 2D shear-sort would require about 1,310 compare-exchange/routing steps.

We have then improved the order of our proposed algorithm to $O(N^{1/4})$. This modified algorithm outperforms the 2D shear-sort for $N > 4{,}096$ elements. To be precise, in this modified version, the first stage of merging requires $15N^{1/4} + o(N^{1/4})$ steps, the second stage of merging can be done in $39N^{1/4} + o(N^{1/4})$ steps, and the overall time complexity is $58N^{1/4} + o(N^{1/4})$. Moreover, the topology of the MM network also provides us with the flexibility of

1) sorting $N^{1/2}$ independent sets of $N^{1/2}$ elements each, in $4N^{1/4} + o(N^{1/4})$ time and
2) sorting $N^{1/4}$ independent sets of $N^{3/4}$ elements each, in $19N^{1/4} + o(N^{1/4})$ time.

2 Basic Concepts and Terminologies

The Multi-Mesh (MM) network consists of n^2 meshes of size $n \times n$ each, which themselves are again arranged in the form of an $n \times n$ matrix. Each constituent $n \times n$ mesh in this matrix is termed as a *block*. In a 2D mesh, each of the four corner processors is of degree two and the remaining boundary processors are each of degree three. In the MM network, these corner and boundary processors of different blocks are interconnected in a suitable manner as described below, so that the degree of each processor is four for $n > 2$ and the diameter of the network is $2n$. (The proof that the diameter of the MM network is upper bounded by $2n$ was given in [23], and that it is exactly equal to $2n$ has been given in [24].)

Each block can be uniquely identified using two coordinates α and β as $B(\alpha, \beta)$. Each of the n^4 processors can be uniquely identified using a four-tuple of the coordinate values. For example, $P(\alpha, \beta, x, y)$ is a processor lying at the xth row and the yth column of the block $B(\alpha, \beta)$. Each of these four coordinates may assume a value between one and n (both inclusive).

A processor designated as P(α, β, x, y) is connected to its four neighbors given by P(α, β, $x \pm 1$, $y \pm 1$), if they exist, using the intrablock connections. Additional connections termed as *interblock links*, among the boundary/corner processors of different blocks, are given in the following way:

1) P(α, β, 1, y) is connected to P(y, β, n, α), where $1 \le y, \alpha, \beta \le n$. As a special case, for $\alpha = y$, these links connect two processors in the same block. These links are called the *vertical interblock links.*
2) P(α, β, x, 1) is connected to P(α, x, β, n) where, $1 \le x, \alpha, \beta \le n$. For $\beta = x$, these links connect two processors in the same block. These links are called the *horizontal interblock links.*

All of these links are two-way connections. An example of a Multi-Mesh network for $n = 3$ is given in Fig. 1, where all the interblock links are not shown.

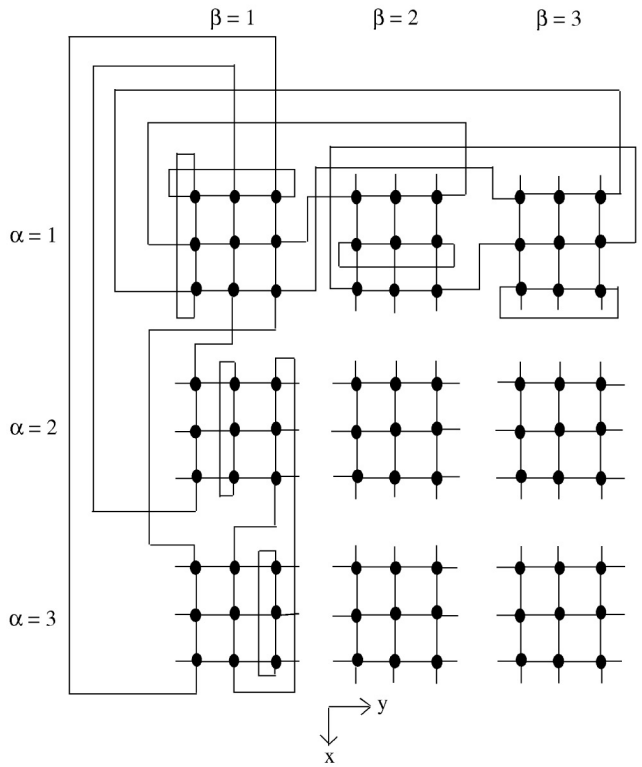

Fig. 1. An example of a Multi-Mesh Network for n = 3. (All interblock links are not shown.)

We refer to the directions corresponding to the coordinate values y, x, α, and β as column, row, third, and fourth dimensions, respectively. Apart from the row (R) and column (C) sort used in a 2D mesh, we introduce the idea of sorting in two more directions—a T-sort for sorting the elements across the third dimension and an F-sort for sorting the elements across the fourth dimension. Let D(α, β, x, y) denote the data element residing in the processors P(α, β, x, y).

DEFINITION 1. *By a C operation, we mean independent column sorts for all the blocks in parallel, where the direction of column sort of all the columns within a block is nondecreasing downward for even values of $\alpha + \beta$ and nondecreasing upward for odd values of $\alpha + \beta$, i.e.,*

$$D(\alpha, \beta, 1, y) \le D(\alpha, \beta, 2, y) \le \cdots \le D(\alpha, \beta, n, y), \text{ if } \alpha + \beta \text{ is even,}$$

and

$$D(\alpha, \beta, 1, y) \ge D(\alpha, \beta, 2, y) \ge \cdots \ge D(\alpha, \beta, n, y), \text{ if } \alpha + \beta \text{ is odd.}$$

DEFINITION 2. *By an R operation, we mean independent row sorts for all the blocks in parallel, where the directions of row sorts of two consecutive rows are opposite within a block (i.e., if the first row is sorted from left to right, then the second row is sorted from right to left and so on); also, the same rows of two consecutive blocks are sorted in opposite directions. In particular, we assume that, when $\alpha + \beta$ is even,*

$$D(\alpha, \beta, x, 1) \le D(\alpha, \beta, x, 2) \le \cdots \le D(\alpha, \beta, x, n), \text{ for odd } x,$$

and

$$D(\alpha, \beta, x, 1) \ge D(\alpha, \beta, x, 2) \ge \cdots \ge D(\alpha, \beta, x, n), \text{ for even } x.$$

When $\alpha + \beta$ is odd, the above inequalities will be reversed.

$\log n$ iterations of such R and C operations followed by an R operation sort the $n \times n$ mesh in a snake-like row-major ordering [19].

Fig. 2a is an example of a 4×4 mesh containing 4^2 (= 16) unordered elements. Fig. 2b is the sorted sequence after shear-sort. The proof of the correctness of the shear-sort algorithm [19] on an $n \times n$ mesh was based on the 0-1 principle [24], with input data elements taken from the set {0, 1} only. The correctness of the proposed algorithm will also be proved using the 0-1 principle and, henceforth, all the input data elements will be considered to be taken from the set {0, 1}. When an input data element is either 0 or 1, the $n \times n$ mesh sorted in snake-like ordering will appear as given in Fig. 2c. We shall use the concept of "clean" and "dirty" rows as mentioned in [18], [19]. Let us denote a "clean" row containing all 1s by ω, a "clean" row containing all 0s by ϕ, and a "dirty" row containing 0s and 1s by δ. Hence, in terms of ϕ, δ, and ωs, a sorted block can be represented by a column of ϕ, δ, and ωs containing, at most, only one δ. In general, 0s and 1s may be intermixed in any order in a dirty row. But, in a sorted block, 0s and 1s in a dirty row are arranged as a sequence of consecutive 0s followed by consecutive 1s. In other words, in a dirty row of a sorted $n \times n$ mesh, the data value changes from 0 to 1 (for increasing order) only once. We denote a dirty row in which the data value changes from 0 to 1 in the left to right direction (e.g., ...00011...) by the symbol δ^+, whereas the dirty rows where the ordering is in the opposite direction (e.g., ...11100...) will be denoted by δ^-.

1	4	9	13
2	6	10	14
3	7	11	15
5	8	12	16

(a)

1	2	3	4
8	7	6	5
9	10	11	12
16	15	14	13

(b)

0	0	0	0
0	0	0	0
0	1	1	1
1	1	1	1

(c)

Fig. 2. Shear sort (a) unordered elements in an n × n mesh, (b) a snake-like row-major sorted sequence, (c) sorted sequence with elements from {0, 1}.

DEFINITION 3. *An ordered block B(α, β) is defined as an even block, where $\alpha + \beta$ is some even number. In such a block, the snake-like sorted sequence starts from the leftmost end of the first row and ends at the nth row.*

DEFINITION 4. *An ordered block B(α, β) is defined as an odd block, where $\alpha + \beta$ is some odd number. In such a block, the snake-like sorted sequence starts from the nth row and ends at the leftmost end of the first row.*

Consider n^3 data elements, denoted by D(*, β, *, *), residing at n blocks (i.e., $n \times n$ meshes) for a fixed value of β, where "*" indicates

all possible values from one to n. If we consider a set of all the data elements for a given (x, y) value from n such blocks, then there will be n^2 such sets of n data elements each, i.e., each set consists of data elements $D(*, \beta, x, y)$ for a fixed value of β, x, and y each.

DEFINITION 5. *A T operation sorts each set of n data elements* $D(*, \beta, x, y)$ *in parallel over the third dimension, so that,*

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

and

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

Note that there is no direct link among the respective processors in the MM network to effect the T operation. Hence, the T operation is accomplished in three stages. The first stage consists of n shifts of data elements along the vertical interblock links (as shown in Fig. 1), so that the ith columns of the blocks $B(*, \beta)$, for a given β, i.e., of all the blocks $B(\alpha, \beta)$, $1 \le \alpha \le n$, are brought to the block $B(i, \beta)$. The situation is explained in Fig. 3 with the help of an example for $n = 3$. Data elements A_1, A_2, and A_3 originally resident on the blocks B(1, 1), B(2, 1), and B(3, 1) will now appear in the first row of block B(1, 1). Similarly, B_1, B_2, and B_3 are brought to row 2 of block B(1, 1), and so on. The T operation actually involves the sorting of the sequences (A_1, A_2, A_3), (B_1, B_2, B_3), etc., each of which is now available in a single row. In the second stage, all the rows in each block for a particular β will be sorted in the same direction (this is different from the R operation of Definition 2, where the consecutive rows of the same block have been sorted in different directions), but the direction of row sorts will alternate for consecutive β values. The third stage is just the reverse of the first stage, in which the sorted data elements are transferred back to the ith columns of the respective blocks having the same β value. In general, the first and the third stages of T operation require a total of $n + n = 2n$ routing steps. The second stage can be completed in n parallel steps of odd-even transposition sort. Thus, the T operation needs a total of $3n$ steps.

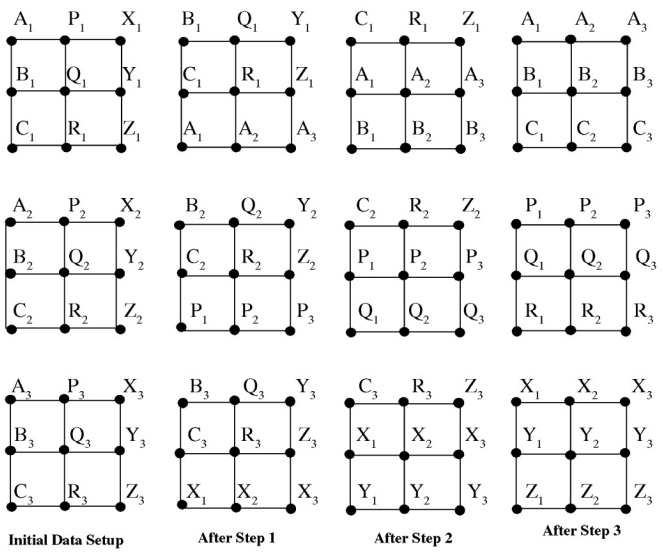

Fig. 3. Three steps of data movements in the first stage of T operation.

DEFINITION 6. *Consider the n elements* $D(\alpha, *, x, y)$ *for a given set of values of* α, x, y. *An* F *operation sorts each such set of n data elements in parallel over the fourth dimension, so that*

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

Again, there is no direct link among the processors whose data elements are to be sorted using F operation. Hence, the F operation is also accomplished in three stages, just like the T operation in a total of $3n$ steps.

DEFINITION 7. *A 3D block is a sequence of alternate odd and even sorted blocks for a given value of* β, *such that*

1) *for odd* β, *all the elements of the sorted block* $B(\alpha, \beta)$ *are less than or equal to all the elements of the sorted block* $B(\alpha + 1, \beta)$, *for* $1 \le \alpha < n$, *and*
2) *for even* β, *the ordering sequence will be just the reverse.*

We call it a block-major snake-like ordering. Thus, in a block-major snake-like ordering,

$$D(1, \beta, *, *) \le D(2, \beta, *, *) \le \cdots \le D(n, \beta, *, *), \text{ for odd values of } \beta$$

and

$$D(1, \beta, *, *) \ge D(2, \beta, *, *) \ge \cdots \ge D(n, \beta, *, *), \text{ for even values of } \beta.$$

DEFINITION 8. *A 4D Block is a sorted sequence of* n^4 *elements consisting of n consecutive 3D blocks, where all the elements of the first 3D block are less than or equal to all the elements of the second 3D block, all the elements of the second 3D block are less than or equal to all the elements of the third 3D block, and so on, i.e.,*

$$D(*, 1, *, *) \le D(*, 2, *, *) \le \cdots \le D(*, n, *, *).$$

Fig. 4 shows an example of a 4D block for $n = 3$, along with its constituent 3D blocks.

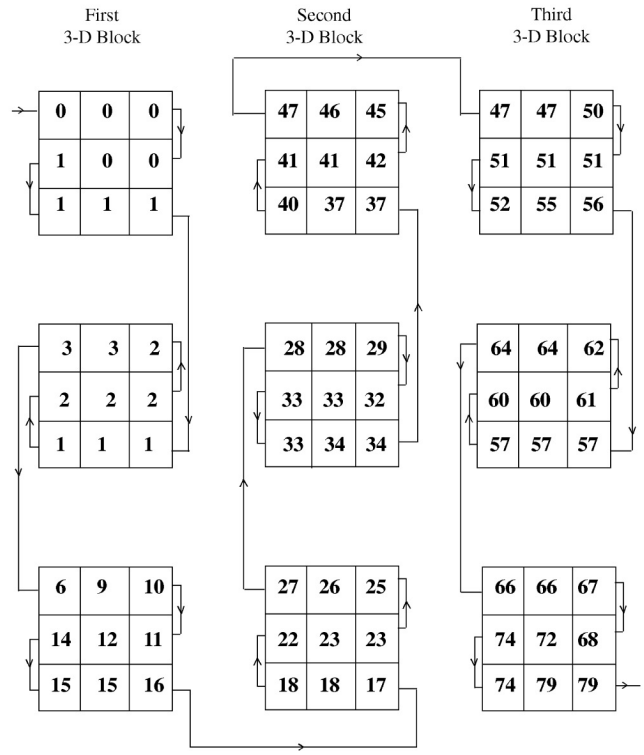

Fig. 4. The final order of elements in a 4D block.

The underlying principle of sorting n^4 elements is the same as column shearing of an $n \times n$ mesh as used in the shear-sort algorithm [19]. First, the proposed algorithm will sort each individual $n \times n$ mesh in parallel using shear-sort in a snake-like row-major ordering. There are n^2 such meshes or blocks. In the second step, we shall merge n such blocks corresponding to a given β-value to make a single 3D block. There will be n such 3D blocks. If we

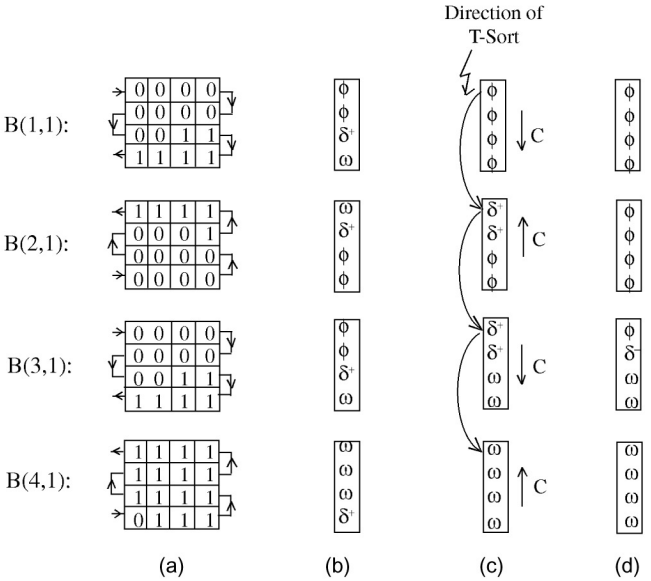

Fig. 5. (a) sorted blocks of 0s and 1s, (b) sorted blocks in terms of ϕ, δ, and ω, (c) blocks after log n steps of T-C operations, (d) blocks in terms of ϕ, δ, and ω in a 3D block.

imagine n rows of a sorted block as n sorted elements, then merging n such blocks will be similar to the merging of n sorted rows in a 2D mesh. In a sense, we are now extending the notion of an element, i.e., one row of a sorted block is now considered to be one single element taken from the set $\{\phi, \delta, \omega\}$ with the ordering $\phi < \delta < \omega$. A sorted 2D block will now appear as $(\phi\phi \cdots \phi\delta\omega \cdots \omega)^T$, where T denotes the transpose. Also, consecutive 2D blocks will be sorted in alternate directions in terms of ϕ, δ ,and ω. An example of four individually sorted 2D blocks of size 4×4 with elements from the set {0, 1} and its equivalent representation in terms of ϕ, δ, and ω have been shown in Figs. 5a and 5b, respectively, for any odd α.

With this *extended* notion of elements, the T-sort on the third dimension will be equivalent to the column sort [19] on a 2D mesh. Also, note that the corresponding columns in two consecutive blocks are sorted in alternate directions. Therefore, for merging n 2D blocks for a given β-value, we need *log n* iterations of T-C operations, after which the resulting structure of the blocks B(1, 1), B(2, 1), B(3, 1), and B(4, 1) is shown in Fig. 5c. Since there are n number of δs and they may be distributed over, at most, two blocks, we need $4n + o(n)$ steps to completely sort these two blocks. We will then show that a sequence of T-C-R-C-R operations will generate a 3D block containing only one δ. Corresponding to the example of Fig. 5a, the resulting structure of blocks B(1, 1), B(2, 1), B(3, 1), and B(4, 1) after these steps is shown in Fig. 5d. Note the increase in the number of ϕs in Fig. 5d from that in Fig. 5c after the δs in Fig. 5c are further sorted.

The final step is to merge these n 3D blocks into a single 4D block. For that, we further extend our notion of an element by calling a column of n ϕs as Φ, a column of n ωs as Ω, and a column of n elements taken from the set $\{\phi, \delta, \omega\}$ as Δ. Thus, the blocks in Fig. 5d can be represented as Φ, Φ, Δ, Ω, respectively. In general, we assume that ϕs, δs, and ωs may be intermixed in any order in a Δ. But, in a 3D block, the only dirty block Δ contains a sequence of consecutive ϕs followed by a single δ (either δ^+ or δ^-) and, finally, a sequence of consecutive ωs. We can now consider a 3D block as a column of elements taken from the set $\{\Phi, \Delta, \Omega\}$ with the ordering $\Phi < \Delta < \Omega$. Merging n such sorted 3D blocks will then be similar to the merging of n sorted rows in a 2D mesh containing these further *extended* elements. The F operation will now be equivalent to a column sort in a 2D mesh and a T operation acts like a row sort in a 2D mesh. We show later that we need *log n* iterations of F-T operations, followed by a fixed sequence of operations for merging two 3D blocks, so as to generate a 4D block. The sorting algorithm is described in the next section.

3 Algorithm for Sorting n^4 Data Elements

3.1 The Algorithm Multi-Sort

Step 1: 2D SORT: (Sorting over row and column dimensions only) Sort each of the n^2 blocks independently in parallel in row-major snake-like ordering, using the optimal algorithm in [19], so that two consecutive blocks in any row or any column of processor blocks are sorted in reverse directions.

Step 2: 3D SORT: (Sorting over row, column and the third dimension) All the n sorted blocks in a column of processor blocks, designated as B(i, β), $1 \le i \le n$, are merged to produce one sorted 3D block, consisting of n^3 sorted elements, in the block-major snake-like ordering. For this merging, perform *log n* iterations of T-C operations, followed by a 2D sort and a sequence of T-C-R-C-R operations. Consecutive 3D blocks are sorted in reverse directions.

REMARK. *The above steps of the algorithm are quite distinct from the 3D sort described in* [7] *which is based on sorting in all the three planes—RC, CT, and TR. On the other hand, the algorithm Multi-Sort does not require sorting in the TR plane.*

Step 3: 4D SORT: (Sorting over all the four dimensions) Merge the n 3D blocks to produce a 4D block of n^4 data elements. For this merging, perform *log n* iterations of F-T operations followed by Steps 1 and 2, and then a sequence of F-T-C-R-T-C-R-C-R operations.

To prove the correctness of the algorithm Multi-Sort, we proceed as follows:

After executing Step 1 of the above algorithm, there will be at most n^2 δs in n^2 different blocks. We now show that there will be at most one δ in any 3D block after the completion of Step 2, and, finally, after executing Step 3, we will be left out with at most only one δ.

LEMMA 1. *The n sorted columns of ϕ, δ, and ωs corresponding to n sorted blocks in a column of processor blocks, can be sorted to form a 3D block after log n iterations of T-C operations, where the δs may be distributed over at most two columns.*

PROOF. The proof is similar to that given in [19] for sorting two dimensional mesh by *log n* iterations of column and row sort. Referring to Fig. 5c, the direction of sorting of ϕ, δ, and ωs (in a column) alternates in consecutive blocks of a 3D block. Hence, the C operation plays the same role as that of a row sort in a 2D mesh. But, the T sort within a 3D block sorts the elements in increasing order, and, hence, it plays the same role as that of the column sort of 2D shear sort. Hence, after *log n* iterations of T-C operations, all the ϕ, δ, and ωs will be sorted in increasing order, so that the δs (which are at most n in number) will be distributed either over a full block or over two consecutive blocks. □

To sort the two blocks containing δs individually, we need a 2D sort leaving at most two δs in the two blocks. Then, a T operation will clean one of the blocks (note that two δs facing each other in consecutive blocks will appear as δ^- and δ^+, respectively, and, hence, will be reduced to just one δ after the T operation) and after that, only one block may contain at most two δs. Next, a sequence of C-R-C will reduce these two δs to at most one δ, where the latter C should span only over two consecutive rows, and, finally, an R operation sorts the corresponding row. In view of these arguments, we have the following theorem.

THEOREM 1. *log n iterations of T-C operations followed by a 2D sort and a sequence of T-C-R-C-R operations merge all the n sorted blocks existing in a column of processor blocks to form a single sorted chain, i.e., a 3D block.*

LEMMA 2. *log n iterations of F-T operations over n 3D blocks will result in a sorted sequence of* Φ, Δ, *and* Ω, *where the* Δ*s may be distributed over, at most, two 3D blocks.*

PROOF. Similar to Lemma 1. Here, instead of ϕ, δ, ω, we will take Φ, Δ, and Ω as elements. Since 3D blocks are sorted in alternate directions and, in the 4D block, all the elements will be in nondecreasing order, F and T operations are equivalent to the column and row operations, respectively, of the 2D sort. □

To sort the two 3D blocks containing the Δs individually, we first use a 2D sort, followed by a 3D sort, so that there will be only one Δ in each 3D block and each such Δ may contain at most one δ. Then, an F operation will clean one of the 3D blocks. A T operation will bring the two Δs in consecutive positions. A C operation followed by an R operation will arrange them appropriately so that the next T operation will reduce these two consecutive Δs to one. Note that, in this single Δ, there may be, at most, two δs, which will be reduced to only one δ by C-R-C operation, and, finally, an R operation will sort this δ.

In view of these arguments, we have the following theorem.

THEOREM 2. *log n iterations of F-T operations followed by a 3D sort and a sequence of* F-T-C-R-T-C-R-C-R *operations merge all the n sorted 3D blocks to form a single sorted chain, i.e., a 4D block.*

3.2 Timing Analysis

Step 1 can be completed in $4n + o(n)$ time. Exploiting the fact that the span of T operation is halved in each iteration of Step 2, the actual time taken for *log n* iterations of T-C will be equal to $3n \log n + 2n - 1$, where we assume that compare-exchange of two elements requires one unit of time. Noting that the T operation following the 2D sort needs to span over only two consecutive processor blocks, Step 2 can be computed in $(3n \log n + 2n - 1)$ (for *log n* iterations of TC) + $(4n + o(n))$ (for 2D sort) + $(2n + 2)$ (for T operation) + $(n + n + 2 + n)$ (for C-R-C-R) = $(3n \log n + 11n + 3)$ time. Similarly, step 3 is accomplished in $(5n \log n + 2n - 1)$ (for *log n* iterations of FT) + $(4n + o(n))$ (for 2D sort) + $(3n \log n + 11n + 3)$ (for 3D sort) + $(2n + 2)$ (for F operation) + $3n$ (for T operation) + $(n + n + (2n + 2) + n + n + 2 + n)$ (for C-R-T-C-R-C-R) = $8n \log n + 29n + 8$ time. The overall time complexity of the above algorithm is, therefore, $11 n \log n + O(n)$; to be precise, the more accurate time complexity comes out to be $11n \log n + 44n + o(n) = 2.75 N^{1/4} \log N + O(N^{1/4})$, where $N = n^4$ is the total number of elements to be sorted.

The above discussions lead to the following theorem.

THEOREM 3. *Algorithm Multi-Sort sorts* $N = n^4$ *elements in* $2.75 N^{1/4} \log N + O(N^{1/4})$ *time.*

REMARK. *If we assume that each compare-exchange and routing step take the same amount of time, then, for* $N = 2^{12}$ *and* 2^{16} *elements, the algorithm Multi-Sort needs roughly* 1,070 *and* 2,270 *compare-exchange/routing steps, respectively. This may be compared with* 1,310 *and* 4,740 *compare-exchange/routing steps for sorting* 2^{12} *and* 2^{16} *elements, respectively, using a 2D shear-sort having time complexity* $4N^{1/2} + O(N^{3/8} \log N)$.

4 Critical Analysis of Data Movements

We can further improve upon the order of the time complexity for sorting by a slight modification of our algorithm. We observe, from Fig. 3, that, after stage 1 of the T operation, all the elements in a TC plane will appear in a single 2D block. If we now modify stage 2 of the T operation as complete sorting of each individual 2D block, then the purpose of *log n* iterations of T-C operations, as indicated in Lemma 1, can really be achieved in n (stage 1)+ $(4n + o(n))$ (modified stage 2) + n (stage 3) steps = $6n + o(n)$ steps. The subsequent 2D sort and T-C-R-C-R operations require $(4n + o(n)) + (2n + 2) + n + n + 2 + n = 9n + 4 + o(n)$ time. Therefore, the total time required for 3D sort is $15n + o(n)$.

Similarly, *log n* iterations of F-T operations can be replaced as follows: First, we perform the stage 1 of the T operation followed by stage 1 of the F operation. It can be verified that, after these two stages, all the elements in an FT plane will appear in a single 2D block. We now sort each individual 2D block, followed by stage 3 of both T and F operations. Thus, the effect of *log n* iterations of F-T operations can be achieved in n (stage 1 of T operation) + n (stage 1 of F operation) + $4n + o(n)$ (modified stage 2 of F and T operations) + n (stage 3 of F operation) + n (stage 3 of T operation) steps = $8n + o(n)$ steps. Hence, the total time required for the modified version of step 3 of the algorithm Multi-Sort turns out to be $(8n + o(n)) + (4n + o(n)) + (15n + o(n)) + (2n + 2)$ (for F operation) + $(3n + n + n + (2n + 2) + n + n + 2 + n)$ (for T-C-R-T-C-R-C-R) = $39n + o(n)$. The overall time complexity of the modified algorithm, therefore, reduces to $58n + o(n)$.

We now state the following theorem.

THEOREM 4. N $(= n^4)$ *elements can be sorted on the Multi-Mesh network in a total of* $58N^{1/4} + o(N^{1/4})$ *comparison-exchange and routing steps.*

For $N = 2^{16}$ elements, this modified Multi-Sort algorithm requires 2,652 compare-exchange/routing steps, whereas the 2D shear-sort takes 4,740 compare-exchange/routing steps to sort the same number of elements. In general, this modified Multi-Sort algorithm outperforms the 2D shear-sort for $N > 4{,}096$ elements.

5 Conclusion

We have proposed an algorithm for sorting N elements on the Multi-Mesh topology in $O(N^{1/4} \log N)$ comparison-exchange/routing steps and then improved its time further to $O(N^{1/4})$ steps. This time complexity may be compared with the $4N^{1/2} + O(N^{3/8} \log N)$ comparison-exchange steps and $O(N^{3/8})$ routing steps in the shear-sort algorithm [19] on a 2D mesh, having almost the same number of links as that of the Multi-Mesh topology. Further investigations are being carried out to reduce the large constant associated with the $O(N^{1/4})$ time complexity of the improved version of our algorithm. It can also be worth investigating to find an appropriate mapping of the Leighton's *Column-Sort* algorithm [7] on the MM network.

Acknowledgments

A preliminary version of this work appeared in the Proceedings of the International Conference on High-Performance Computing (HiPC '95), pp. 707-712, New Delhi, India, December 27-30, 1995.

References

[1] R. Reischuk, "A Fast Probabilistic Parallel Sorting Algorithm," *Proc. 22nd IEEE Symp. FOCS*, pp. 212-219, 1981.
[2] R. Cole, "Parallel Merge Sort," *Proc. 27th IEEE Symp. FOCS*, pp. 511-516, 1986.
[3] S.G. Akl and N. Santoro, "Optimal Parallel Merging and Sorting without Memory Conflicts," *IEEE Trans. Computers*, vol. 36, no. 11, pp. 1,367-1,369, Nov. 1987.

[4] M. Ajatai, J. Komlos, and E. Szemeredi, "An $O(n \log n)$ Sorting Network," *Proc. 15th ACM STOC*, pp. 1-9, 1983.
[5] F.T. Leighton, "Tight Bounds on the Complexity of Parallel Sorting," *IEEE Trans. Computers*, vol. 34, no. 4, pp. 344-354, Apr. 1985.
[6] K. Batcher, "Sorting Networks and Their Applications," *Proc. AFIPS Spring Joint Computing Conf.*, vol. 32, pp. 307-314, 1968.
[7] F.T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*. San Mateo, Calif.: Morgan Kaufmann, 1992.
[8] F. Preparata, "New Parallel Sorting Schemes," *IEEE Trans. Computers*, vol. 27, no. 7, pp. 669-673, July 1978.
[9] D. Nassimi and S. Sahni, "Parallel Permutation and Sorting Algorithms and a New Generalized Connection Network," *J. ACM*, vol. 29, no. 3, pp. 642-667, 1982.
[10] A. Aggarwal and M.-D. Huang, "Network Complexity of Sorting and Graph Problems and Simulating CRCW PRAMs by Interconnection Networks," *Lecture Notes on Computer Science*, vol. 319, pp. 339-350. Springer-Verlag, July 1988.
[11] R. Cypher and J.L.C. Sanz, "Optimal Sorting on Reduced Architectures," *Proc. ICPP*, vol. 3, pp. 308-311, Pennsylvania State Univ., Aug. 1988.
[12] G. Plaxton, "On the Network Complexity of Selection," *Proc. 30th Ann. Symp. FOCS*, pp. 396-401, Nov. 1989.
[13] R. Cypher and G. Plaxton, "Deterministic Sorting in Nearly Logarithmic Time on the Hypercube and Related Computers," *Proc. 22nd Ann. ACM Symp. Theory of Computing*, pp. 193-203, 1990.
[14] J. Reif and L. Valiant, "A Logarithmic Time Sort for Linear Size Networks," *J. ACM*, vol. 34, no. 1, pp. 60-76, 1987.
[15] T. Leighton, B. Maggs, A. Ranade, and S. Rao, "Randomized Routing and Sorting in Fixed-Connection Networks," *J. Algorithms*, vol. 17, no. 1, pp. 157-205, 1994.
[16] T. Leighton and G. Plaxton, "A (Fairly) Simple Circuit that (Usually) Sorts," *Proc. 31st Ann. Symp. FOCS*, pp. 264-274, Oct. 1990.
[17] C.D. Thompson and H.T. Kung, "Sorting on a Mesh-Connected Parallel Computer," *Comm. ACM*, vol. 20, pp. 263-271, Apr. 1977.
[18] C.P. Schnorr and A. Shamir, "An Optimal Sorting Algorithm for Mesh Connected Computers," *Proc. 18th ACM STOC*, pp. 255-263, May 1986.
[19] I.D. Scherson and S. Sen, "Parallel Sorting in Two-Dimensional VLSI Models of Computation," *IEEE Trans. Computers*, vol. 38, no. 2, pp. 238-249, Feb. 1989.
[20] M. Kunde, "Optimal Sorting on Multidimensionally Mesh-Connected Arrays," *Proc. Fourth Ann. STACS '87, Lecture Notes in Computer Science*, vol. 247, pp. 408-419, 1987.
[21] M. Kunde, "Routing and Sorting on Mesh-Connected Arrays," *Proc. Third Aegean Workshop Computing, Lecture Notes in Computer Science*, vol. 319, pp. 423-433, 1988.
[22] P.F. Corbett and I.D. Scherson, "Sorting in Mesh Connected Multiprocessors," *IEEE Trans. Parallel and Distributed Systems*, vol. 3, no. 5, pp. 626-632, Sept. 1992.
[23] D. Das and B.P. Sinha, "Multi-Mesh—An Efficient Topology for Parallel Processing," *Proc. Ninth Int'l Parallel Processing Symp.*, pp. 17-21, Santa Barbara, Calif., Apr. 25-28, 1995.
[24] D. Das, M. De, and B.P. Sinha, "A New Network Topology with Multiple Meshes," submitted to *IEEE Trans. Computers*.
[25] D.E. Knuth, *The Art of Computer Programming, Sorting and Searching*, vol. 3. Reading Mass.: Addison-Wesley, 1973.