J Supercomput (2011) 57:276–313
DOI 10.1007/s11227-010-0404-4

Analysis of Multi-Sort Algorithm on Multi-Mesh of Trees (MMT) architecture

Nitin Rakesh · Nitin

Published online: 11 March 2010

Abstract Various sorting algorithms using parallel architectures have been proposed in the search for more efficient results. This paper introduces the Multi-Sort Algorithm for Multi-Mesh of Trees (MMT) Architecture for $N = n^4$ elements with more efficient time complexity compared to previous architectures. The shear sort algorithm on Single Instruction Multiple Data (SIMD) mesh model requires $4\sqrt{N} + O\sqrt{N}$ time for sorting N elements, arranged on a $\sqrt{N} \times \sqrt{N}$ mesh, whereas Multi-Sort algorithm on the SIMD Multi-Mesh (MM) Architecture takes $O(N^{1/4})$ time for sorting the same N elements, which proves that Multi-Sort is a better sorting approach. We have improved the time complexity of intrablock Sort. The Communication time complexity for 2D Sort in MM is $O(n)$, whereas this time in MMT is $O(\log n)$. The time complexity of compare–exchange step in MMT is same as that in MM, i.e., $O(n)$. It has been found that the time complexity of the Multi-Sort on MMT has been improved as on Multi-Mesh architecture.

Keywords MMT · Multi-Sort · Communication time · Compare exchange

N. Rakesh, Member, IEEE.
Nitin, Senior Member, IACSIT and Member, SIAM and IEEE.

N. Rakesh · Nitin (✉)
Department of Computer Science & Engineering and Information Technology, Jaypee University of Information Technology, Waknaghat, Solan 173215, Himachal Pradesh, India
e-mail: delnitin@juit.ac.in

N. Rakesh
e-mail: nitin.rakesh@juit.ac.in

Nitin
e-mail: delnitin@ufl.edu

1 Introduction and motivation

Several interconnection networks based on the mesh topology have been developed as parallel processing systems. In search of faster algorithms a network topology, called Multi-Mesh of Trees (MMT) [1], which is a hybrid product of multi-mesh and mesh of trees networks was recently proposed. An $n \times n$ MMT network can be built using n^2 mesh of trees, each of size $n \times n$, and therefore it has n^4 processors in total. The network has the same bisection width of $2n(n-1)$ and same number of edges, i.e., $2n^4$, as that of Multi-Mesh (MM) [2–4]. The diameter of the MMT network is found to be $4\log n + 2$. This can be compared with the diameter of the multi-mesh network, which is $2n$. Due to efficient topological properties of MMT architecture over MM network [1] (as shown in Fig. 1(a)), this has given us a strong base to propose algorithms (Multi-Sort is one of them) on this architecture, and this is the source of motivation for this research.

There are basically two types of links that are available in the MMT architecture. One type is within the block connecting different processors, called intrablock links, and the other type is interblock links that connect one block to another block. Only the boundary and the corner processor of one block can communicate with the other boundary processor of another block. Inter and intrablock links are again classified into two kinds: one is horizontal and the other is vertical. Thus, the topological property of MMT enables implementing several sorting algorithms in a more efficient manner (e.g., 2D Sort has been implemented on both MM and MMT with $O(n)$ on MM and $O(\log n)$ on MMT; a comparison in shown in Fig. 2). This paper is focused

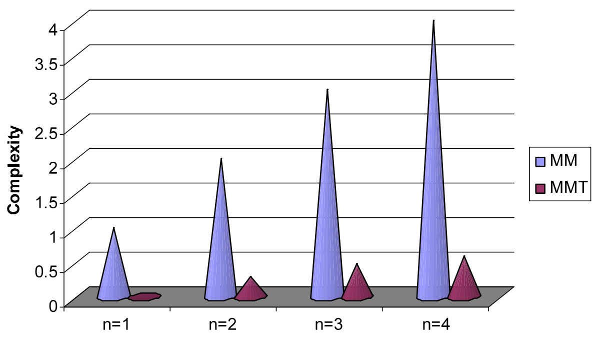


Fig. 1 Comparison of MMT and MM on the basis of communication links, solution of polynomial equations, one-to-all and row & column broadcast

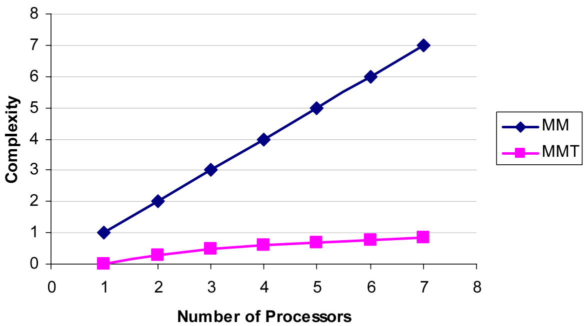


Fig. 2 A comparison between 2D Sort on MM and MMT for different values number of processors

on parallel sorting algorithms [5–8] that are implemented on parallel architecture [9–17].

Specifically, in this paper, we have proposed the Multi-Sort Algorithm on the MMT architecture with more efficient results than MM. The paper is divided into five sections. Section 1 gives the introduction to this paper, the approach used, and the source of motivation for this research. Section 2 describes the proposed algorithm and the operations involved in the algorithms. The definition for each operation is described and explained with the help of a running example (a common example to explain all operations, which is continuing till this section), where some data values are provided to a given set of processors. In Sect. 3, an algorithm for sorting $N = n^4$ data elements is described with the help of an initial data input, and its correctness is proved using a dry run of this algorithm. Section 4 describes the use of the algorithm followed by the conclusion and references.

2 Proposed operations

To propose the Multi-Sort Algorithm on MMT, some basic operations are used that contribute in the efficient implementation of the algorithms. These operations are defined and explained in this section; for a better explanation, a common example is used with some common number of processors.

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

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

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

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 the block (i.e., if the first row is sorted from left to right, the second row is sorted 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,

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

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

when $\alpha + \beta$ is odd,

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

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

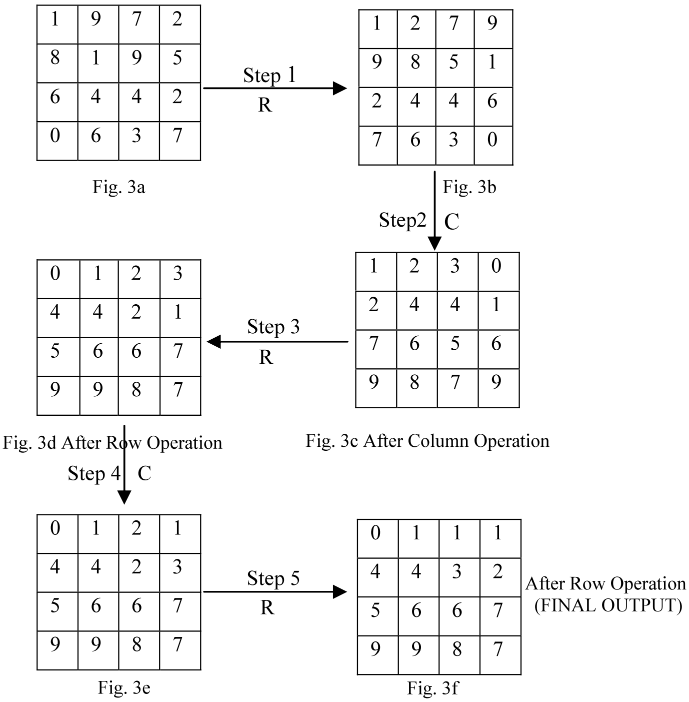


Fig. 3 An R–C operation

$\log n$ iterations of such R and C operations followed by an R operation sort the $n \times n$ mesh in the snake-like row major ordering. Figure 3 gives an example of a 4×4 mesh containing 4^2 $(= 16)$ unsorted elements. Figure 3(f) is the sorted sequence after shear sort. Basically, for our architecture we are relying on shear sort for sorting elements within a block. The proof of the correctness of the shear sort algorithm on an $n \times n$ mesh was based on the 0–1 principle, with the input elements taken from the set $\{0, 1\}$ only.

Let us denote a clean row containing all 1s by ϖ, a clean row containing all 0s by ϕ, and a dirty row containing 0s and 1s by δ. Hence, in terms of ϖ, ϕ, δs can be used for the clean and dirty rows. A sorted block can be represented by a column of ϖ, ϕ, and δs, containing at most only one δ. In general, 0s and 1s can 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), and 1 to 0 (for decreasing 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. $\ldots 0011 \ldots$) by the symbol δ^+, whereas the dirty row in which the ordering is in the opposite direction (i.e., $\ldots 111000 \ldots$) will be denoted by δ^-. A typical example of the shear sort can be shown as Fig. 3.

Similarly, this shear sort can also be implemented on the input value set $\{0,1\}$. A typical example of the sorted block can be (see Fig. 4):

Fig. 4 Sorted sequence with elements from {0, 1}

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

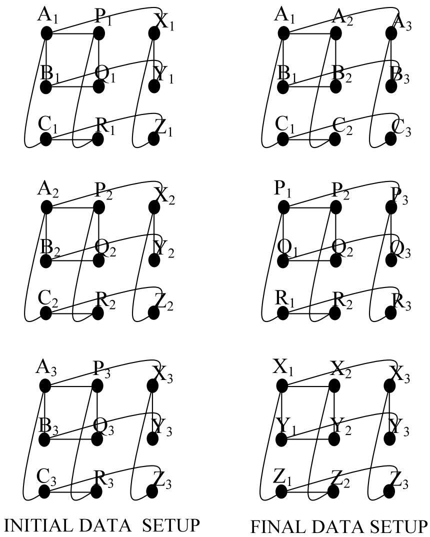


Fig. 5 Position of data elements before and after the T operation

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

Definition 4 An ordered block $B(\alpha, \beta)$ is defined as an odd block, where the $\alpha + \beta$ is some odd number. In such a block, the snake-like sorted sequence starts from the nth row ends at the leftmost end of the first row. Consider n^3 data elements, denoted by $D(*, \beta, *, *)$ residing at n blocks (i.e., $n \times n$ meshes) for a fixed value of β. Here "$*$" indicates all possible values from 1 to n. If we consider a set of all the data elements for a given (x, x) 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 fixed values of β, x, and y.

Definition 5 The T operation sorts each set of n data elements $D(*, \beta, x, y)$ in parallel over the third dimension, so that $D(1, \beta, x, y) \leq D(2, \beta, x, y) \leq \cdots \leq D(n, \beta, x, y)$ if β is odd, and $D(1, \beta, x, y) \geq D(2, \beta, x, y) \geq \cdots \geq D(n, \beta, x, y)$ if β is even. Note that there is no direct link among the respective processors in the MMT network to affect 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. 5), so that the ith columns of the blocks $B(*, \beta)$ for a given β, i.e., of all the blocks $B(\alpha, \beta)$, $1 \leq \alpha \leq n$, are brought to the block

$B(i, \beta)$. The situation is explained in Fig. 5 with the help of an example for $n = 3$. Data elements A1, A2, and A3 originally residing at 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, B1, B2, and B3 are brought to row 2 of block $B(1, 1)$, and so on. The T operation actually involves the sorting of the sequences (A1, A2, A3), (B1, B2, B3), 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 are 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 stage requires $\log n + 1$ routing steps and the third stages of T operation require $2 \log n + 1$ steps, hence a total of $3 \log n + 2$ routing steps. The second stage can be completed in n parallel steps of Compare-steps. Thus, the T operation needs a total of $3 \log n + n + 2$ routing steps.

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 $D(\alpha, 1, x, y) \leq D(\alpha, 2, x, y) \leq \cdots \leq D(\alpha, n, x, y)$. Again, there is no direct link among the processors whose data elements are to be sorted using an F operation. Hence, the F operation is also accomplished in three stages. In the first stage, for a particular value of α, i.e., for all values of β where $1 \leq \beta \leq n$, the ith row is brought to the block $B(\alpha, j)$ and the βth row in the block, i.e., the first row of $B(1, 1)$ is brought to the first row of $B(1, 1)$, whereas the first row of block $B(1, 2)$ is brought to the second row of the block $B(1, 1)$, and so on. In the second step, the column sort has to be performed. This column sort is performed in the same direction in all the blocks (i.e., the first row of every block contains least values in that block and the last row contains the largest values). The third step is just the inverse operation of the operations performed in the first step. Just like the T operation, the F operation has $3 \log n + n + 2$ *steps* in total.

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 b, 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 $1 \leq \alpha < n$, and (2) for even b, the ordering sequence is just the reverse. We call it a block-major snake-like ordering. Thus, in a block-major snake-like ordering, $D(1, \beta, *, *) \leq D(2, \beta, *, *) \leq \cdots \leq D(n, \beta, *, *)$, for odd values of β, and $D(1, \beta, *, *) \geq D(2, \beta, *, *) \geq \cdots \geq D(n, \beta, *, *)$, for even values of β.

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

2.1 C operation

A C operation involves three steps; all these steps are defined below:

Step 1. $\forall \alpha, \beta, i, j : 1 \leq \alpha, \beta, i, j \leq n$ do in parallel
$B[\alpha, \beta, 1, j][i, 1] \leftarrow A[\alpha, \beta, i, j]$

Step 2. $\forall \alpha, \beta : 1 \leq \alpha, \beta \leq n$
(a) if $((\alpha + \beta)\ mod\ 2 = 0)$ then
$\forall \alpha, \beta : 1 \leq \alpha, \beta \leq n$ do in parallel
sort_ascend()
(b) else
$\forall \alpha, \beta : 1 \leq \alpha, \beta \leq n$ do in parallel
sort_descend()

Step 3. $\forall \alpha, \beta, i, j : 1 \leq \alpha, \beta, i, j \leq n$ do in parallel
$A[\alpha, \beta, i, j] \leftarrow B[\alpha, \beta, 1, j][i, 1]$

2.1.1 Explanation (C operation)

The C operation is explained below. The initial data elements present with each processor are given as in Fig. 6. Further, all the three steps of C operations are explained with the help of diagrammatic example (as shown in Figs. 7, 8 and 9).

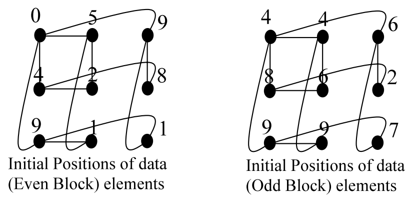


Fig. 6 Stage before a C operation (the initial data elements with even and odd blocks of MMT architecture for $n = 3$ with all processors are shown here)

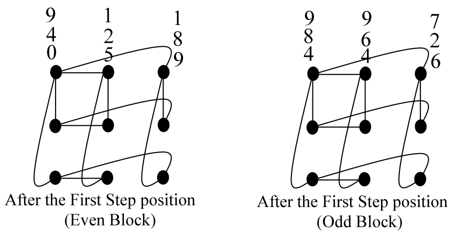


Fig. 7 Position of data elements after implementation of Step 1 of a C operation

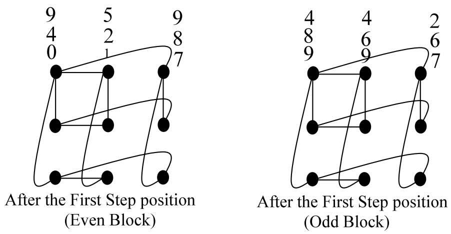


Fig. 8 Position of data elements after implementation of Step 2 of a C operation

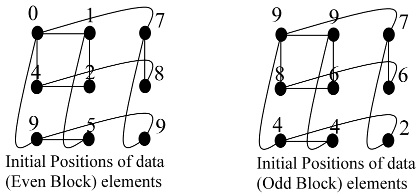


Fig. 9 Position of data elements after completion of Step 3 of a C operation

Step 1. $\forall \alpha, \beta, i, j : 1 \leq \alpha, \beta, i, j \leq n$ do in parallel
$B[\alpha, \beta, 1, j][i, 1] \leftarrow A[\alpha, \beta, i, j]$
This explanation is for the first block (similar work for the entire block is done in parallel):
$B[1, 1, 1, 1][1, 1] \leftarrow A[1, 1, 1, 1] B[1, 1, 1, 1][2, 1] \leftarrow A[1, 1, 2, 1] B[1, 1, 1, 1][3, 1] \leftarrow A[1, 1, 3, 1]$
$B[1, 1, 1, 2][1, 1] \leftarrow A[1, 1, 1, 2] B[1, 1, 1, 2][2, 1] \leftarrow A[1, 1, 2, 2] B[1, 1, 1, 2][3, 1] \leftarrow A[1, 1, 3, 2]$
$B[1, 1, 1, 3][1, 1] \leftarrow A[1, 1, 1, 3] B[1, 1, 1, 3][2, 1] \leftarrow A[1, 1, 2, 3] B[1, 1, 1, 3][3, 1] \leftarrow A[1, 1, 3, 3]$

Step 2. $\forall \alpha, \beta : 1 \leq \alpha, \beta \leq n$
(a) if $((\alpha + \beta)$ *mod* $2 = 0)$ then
$\forall \alpha, \beta : 1 \leq \alpha, \beta \leq n$ do in parallel
sort_ascend(α, β)
(b) else
$\forall \alpha, \beta : 1 \leq \alpha, \beta \leq n$ do in parallel
sort_descend(α, β)

Step 3. $\forall \alpha, \beta, i, j : 1 \leq \alpha, \beta, i, j \leq n$ do in parallel
$A[\alpha, \beta, i, j] \leftarrow B[\alpha, \beta, 1, j][i, 1]$

2.2 R operation

An R operation is performed in three steps; these steps are elaborated further with diagrammatic explanation, implemented on the same initial data elements as in Fig. 6.

Step 1. $\forall \alpha, \beta, i, j : 1 \leq \alpha, \beta, i, j \leq n$ do in parallel
$B[\alpha, \beta, i, 1][1, j] \leftarrow A[\alpha, \beta, i, j]$

Step 2. $\forall \alpha, \beta : 1 \leq \alpha, \beta \leq n$
if $((\alpha + \beta)$ *mod* $2 = 0)$ then
$\forall \alpha, \beta : 1 \leq \alpha, \beta \leq n$ do in parallel
(a) if $(i$ *mod* $2 \neq 0)$ then
$\forall i : 1 \leq i \leq n$ do in parallel
sort_ascend()
(ab) else
$\forall i : 1 \leq i \leq n$ do in parallel
sort_descend();
endif
(b) else
$\forall \alpha, \beta : 1 \leq \alpha, \beta \leq n$ do in parallel
(ba) if $(i$ *mod* $2 \neq 0)$ then
$\forall i : 1 \leq i \leq n$ do in parallel
sort_descend()

(bb) else
$\forall i : 1 \le i \le n$ do in parallel
sort_ascend();
endif
endif

Step 3. $\forall \alpha, \beta, i, j : 1 \le \alpha, \beta, i, j \le n$ do in parallel
$A[\alpha, \beta, i, j] \leftarrow B[\alpha, \beta, i, 1][1, j]$

2.2.1 Explanation (R operation)

An explanation of an R operation is given below, and it is provided using a diagrammatic flow of data elements in the network (as shown in Figs. 11, 12 and 13). The R operation is divided into three steps. The completion of these steps collectively forms the R operation.

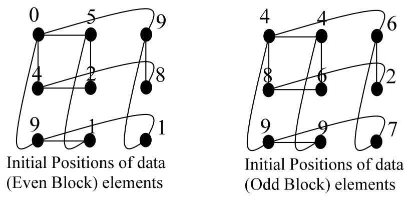


Fig. 10 Stage before an R operation (Fig. 6 is used here, but for implementing an R operation. It is shown again to make the operation on these data elements clearer and more understandable for the reader)

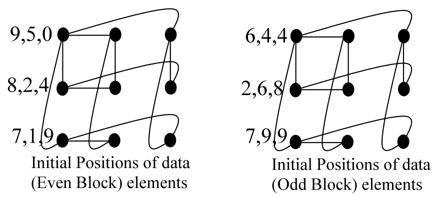


Fig. 11 Positions after Step 1 of an R operation (the figure shows the position of data elements of Fig. 10)

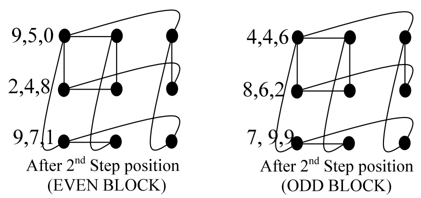


Fig. 12 Position of data elements of Fig. 11 after Step 2 of an R operation

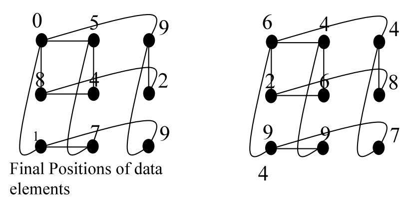


Fig. 13 Position of data elements after implementation of Step 3 of an R operation

Step 1. $\forall \alpha, \beta, i, j : 1 \leq \alpha, \beta, i, j \leq n$ do in parallel
$B[\alpha, \beta, i, 1][1, j] \leftarrow A[\alpha, \beta, i, j]$
This explanation is for the first block (similar work for the entire block is done in parallel):
$B[1, 1, 1, 1][1, 1] \leftarrow A[1, 1, 1, 1] B[1, 1, 1, 1][1, 2] \leftarrow A[1, 1, 1, 2] B[1, 1, 1, 1][1, 3] \leftarrow A[1, 1, 1, 3]$
$B[1, 1, 2, 1][1, 1] \leftarrow A[1, 1, 2, 1] B[1, 1, 2, 1][1, 2] \leftarrow A[1, 1, 2, 2] B[1, 1, 2, 1][1, 3] \leftarrow A[1, 1, 2, 3]$
$B[1, 1, 3, 1][1, 1] \leftarrow A[1, 1, 3, 1] B[1, 1, 3, 1][1, 2] \leftarrow A[1, 1, 3, 2] B[1, 1, 3, 1][1, 3] \leftarrow A[1, 1, 3, 3]$

Step 2. $\forall \alpha, \beta : 1 \leq \alpha, \beta \leq n$
if $((\alpha + \beta)\ mod\ 2 = 0)$ then
$\forall \alpha, \beta : 1 \leq \alpha, \beta \leq n$ do in parallel
(a) if$(i\ mod\ 2 \neq 0)$ then
$\forall i : 1 \leq i \leq n$ do in parallel
sort_ascend()
else
$\forall i : 1 \leq i \leq n$ do in parallel
sort_descend();
endif
(b) else
$\forall \alpha, \beta : 1 \leq \alpha, \beta \leq n$ do in parallel
(ba) if $(i\ mod\ 2 \neq 0)$ then
$\forall i : 1 \leq i \leq n$ do in parallel
sort_descend()
(bb) else
$\forall i : 1 \leq i \leq n$ do in parallel
sort_ascend();
endif
endif

Step 3. $\forall \alpha, \beta, i, j : 1 \leq \alpha, \beta, i, j \leq n$ do in parallel
$A[\alpha, \beta, i, j] \leftarrow B[\alpha, \beta, 1, j][i, 1]$

2.3 T operation

A T operation is a larger operation (compared to C and R operations), which is divided into three stages, i.e., Stage 1 (which consist of four steps), Stage 2 (which consist of one step) and Stage 3 (which consist of five steps). These steps are explained in Figs. 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, stepwise. The initial input of a T operation is shown in Fig. 14.

Stage 1

Step 1. $\forall \alpha, \beta, j : 1 \leq \alpha, \beta, j \leq n$ do in parallel
$B[\alpha, \beta, 1, j][i, 1] \leftarrow A[\alpha, \beta, i, j]$

Step 2. $\forall \alpha, \beta, i, j : 1 \leq \alpha, \beta, j \leq n\ AND\ i = 1$ do in parallel
$C[j, \beta, n, \alpha][i, 1] \leftarrow B[\alpha, \beta, 1, j][i, 1]$

Step 3. $\forall \alpha, \beta, j : 1 \leq \alpha, \beta, j \leq n$ do in parallel
Call Vert_links$(\alpha, \beta, n, j, n - 1)$
$C[\alpha, \beta, k, j][i, 1] \leftarrow C[\alpha, \beta, n, j][i, 1]$

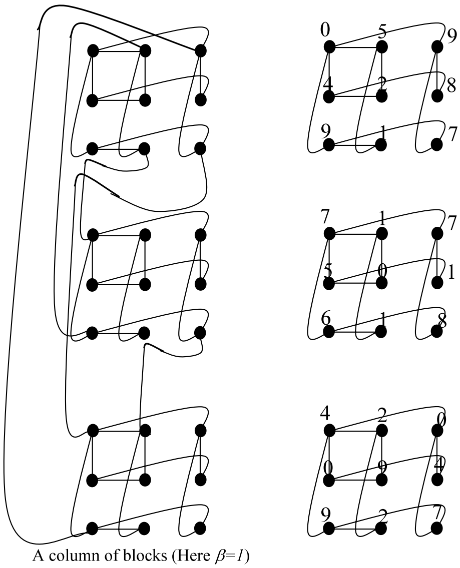


Fig. 14 Initial input of a T operation (the left figure shows the connectivity of processor according to MMT architecture, and the right figure shows initial data elements present with these processors)

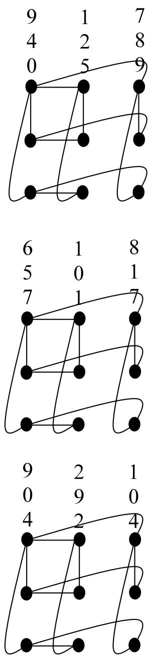


Fig. 15 Positions of data elements after Step 1 (all connections are not shown in the above figure)

Step 4. $\forall \alpha, \beta, i : 1 \leq \alpha, \beta, j \leq n$ do in parallel
$C[\alpha, \beta, i, 1][1, j] \leftarrow C[\alpha, \beta, i, j][i, 1]$

Stage 2

Step 1. Call special_R_operation

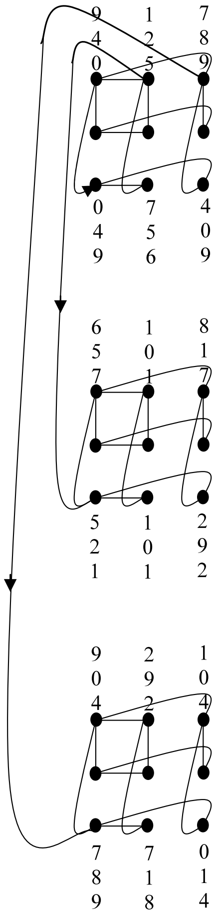


Fig. 16 Position of data elements after Step 2 (all connections are not shown in the above figure)

Stage 3

Step 1. $\forall\alpha, \beta, i : 1 \leq \alpha, \beta, i \leq n$ do in parallel
$C[\alpha, \beta, i, j][i, 1] \leftarrow C[\alpha, \beta, i, 1][1, j]$

Step 2. $\forall\alpha, \beta, j : 1 \leq \alpha, \beta, j \leq n$ do in parallel
$C[\alpha, \beta, 1, j][i, 1] \leftarrow C[\alpha, \beta, i, j][i, 1]$

Step 3. $\forall\alpha, \beta, j : 1 \leq \alpha, \beta, j \leq n$ do in parallel
$C[\alpha, \beta, n, j][i, 1] \leftarrow C[\alpha, \beta, 1, j][i, 1]$

Step 4. $\forall\alpha, \beta, j : 1 \leq \alpha, \beta, j \leq n$ do in parallel
$C[\alpha, \beta, 1, j][i, 1] \leftarrow C[j, \beta, n, \alpha][i, 1]$

Step 5. $\forall\alpha, \beta, j : 1 \leq \alpha, \beta, j \leq n$ do in parallel
$A[\alpha, \beta, i, j] \leftarrow C[\alpha, \beta, 1, j][i, 1]$

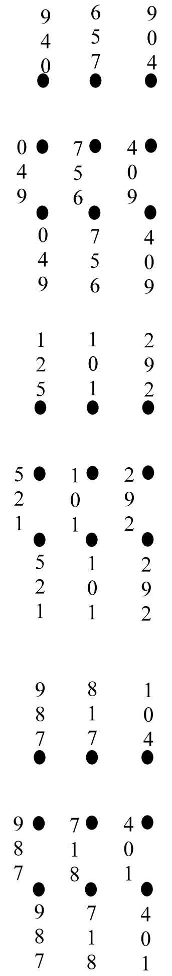


Fig. 17 Step 3 of Stage 1. Positions of other blocks and data elements (connections are not shown in this figure)

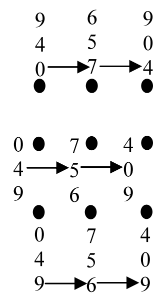


Fig. 18 Step 3 of Stage 1. Step 4 performed on a single block. Similar operation can be performed on all the blocks

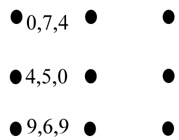


Fig. 19 Step 3 of Stage 1. Step 4 can be performed similarly on all blocks

Fig. 20 Step 1 of Stage 2

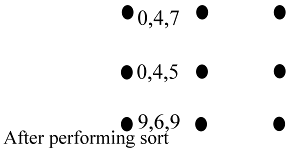


Fig. 21 Step 1 of Stage 3

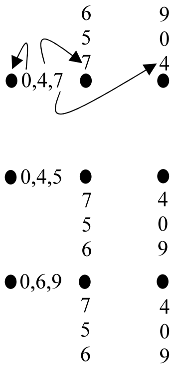


Fig. 22 Position of data elements after Step 1. Step 2 is applied here

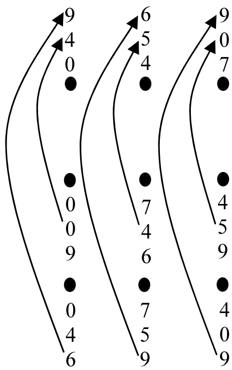
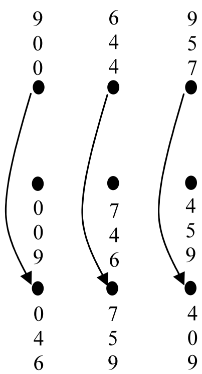


Fig. 23 Position of data elements after Step 2. Step 3 is applied here



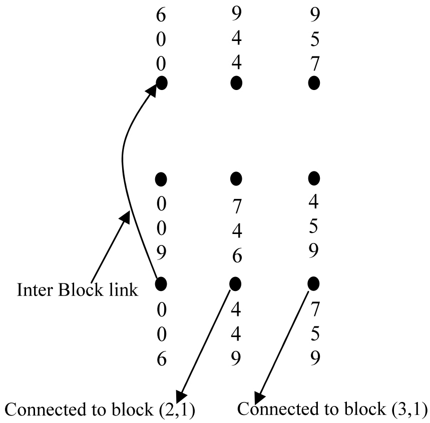


Fig. 24 Position of data elements after Step 3. Step 4 is applied here

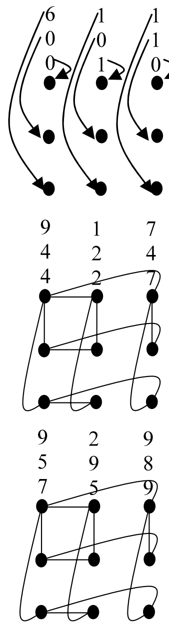


Fig. 25 Step 5

2.3.1 Explanation (T operation)

Stage 1

Step 1. $\forall \alpha, \beta, j : 1 \leq \alpha, \beta, j \leq n$ do in parallel
$B[\alpha, \beta, 1, j][i, 1] \leftarrow A[\alpha, \beta, i, j]$

Step 2. $\forall \alpha, \beta, i, j : 1 \leq \alpha, \beta, j \leq n, i = 1$ do in parallel
$C[j, \beta, n, \alpha][i, 1] \leftarrow B[\alpha, \beta, 1, j][i, 1]$

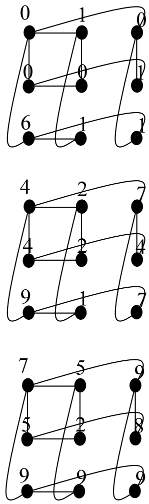


Fig. 26 Position of data elements after Step 5

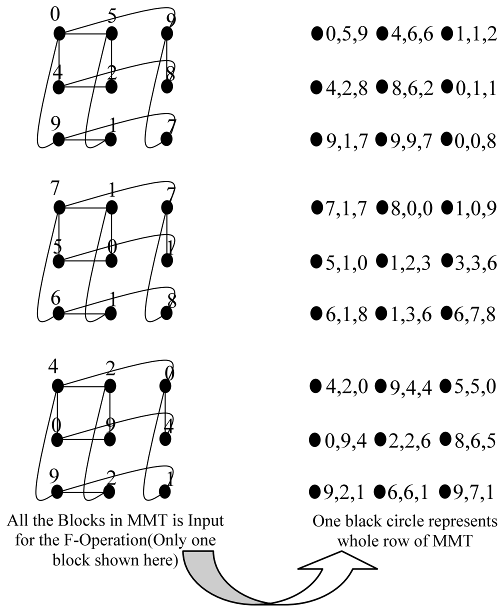


Fig. 27 Step 1 of an F operation

Step 3. $\forall \alpha, \beta, j : 1 \leq \alpha, \beta, j \leq n$ do in parallel
Call Vert_links$(\alpha, \beta, n, j, n-1)$
$C[\alpha, \beta, k, j][i, 1] \leftarrow C[\alpha, \beta, n, j][i, 1]$

Step 4. $\forall \alpha, \beta, I : 1 \leq \alpha, \beta, i \leq n$ do in parallel
$C[\alpha, \beta, i, 1][1, j] \leftarrow C[\alpha, \beta, i, j][i, 1]$

Stage 2

Step 1. Call special_R_operation

Stage 3

Step 1. $\forall \alpha, \beta, i : 1 \leq \alpha, \beta, i \leq n$ do in parallel
$C[\alpha, \beta, i, j][i, 1] \leftarrow C[\alpha, \beta, i, 1][1, j]$

Step 2. $\forall \alpha, \beta, j : 1 \leq \alpha, \beta, j \leq n$ do in parallel
$C[\alpha, \beta, 1, j][i, 1] \leftarrow C[\alpha, \beta, i, j][i, 1]$

Step 3. $\forall \alpha, \beta, j : 1 \leq \alpha, \beta, j \leq n$ do in parallel
$C[\alpha, \beta, n, j][i, 1] \leftarrow C[\alpha, \beta, 1, j][i, 1]$

Step 4. $\forall \alpha, \beta, j : 1 \leq \alpha, \beta, j \leq n$ do in parallel
$C[\alpha, \beta, 1, j][i, 1] \leftarrow C[j, \beta, n, \alpha][i, 1]$

Step 5. $\forall \alpha, \beta, j : 1 \leq \alpha, \beta, j \leq n$ do in parallel
$A[\alpha, \beta, i, j] \leftarrow C[\alpha, \beta, 1, j][i, 1]$

2.4 F operation

An F operation is also divided into three stages. Stage 1 consists of four steps, Stage 2 consists of one step, and Stage 3 consists of five steps. These steps are explained in Figs. 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, stepwise and Fig. 37 shows the final order

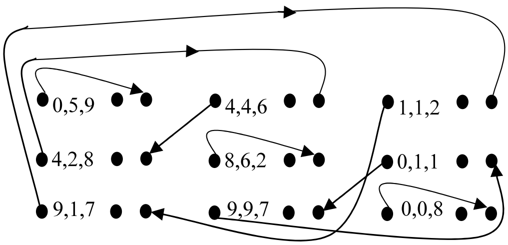


Fig. 28 Positions of data elements after Step 2. (Only the horizontal plane blocks are shown)

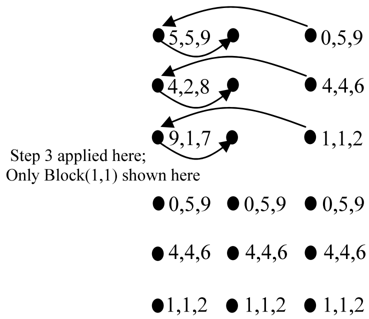


Fig. 29 Position of data elements after Step 3; only block (1,1) is shown

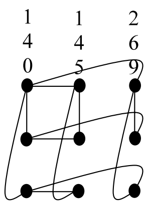


Fig. 30 After Step 4; only block (1,1) is shown here

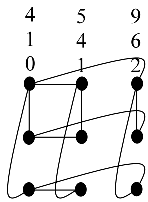


Fig. 31 After Step 1 (Stage 2)

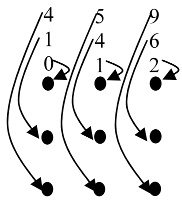


Fig. 32 Stage 3 of Step 1; only block (1,1) is shown

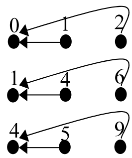


Fig. 33 After Step 2; only block (1,1) is shown here

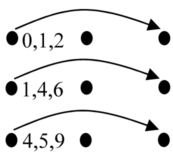


Fig. 34 After Step 3; only block (1,1) is shown here

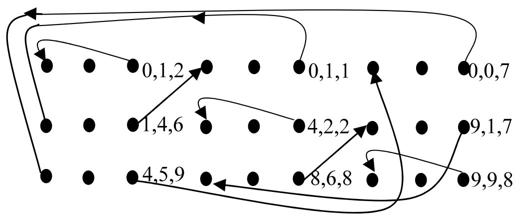


Fig. 35 After Step 4; only the first horizontal plane is shown

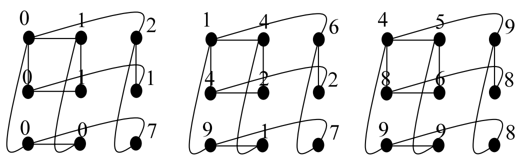


Fig. 36 Positions of the data elements after Step 5

Fig. 37 Final order of data

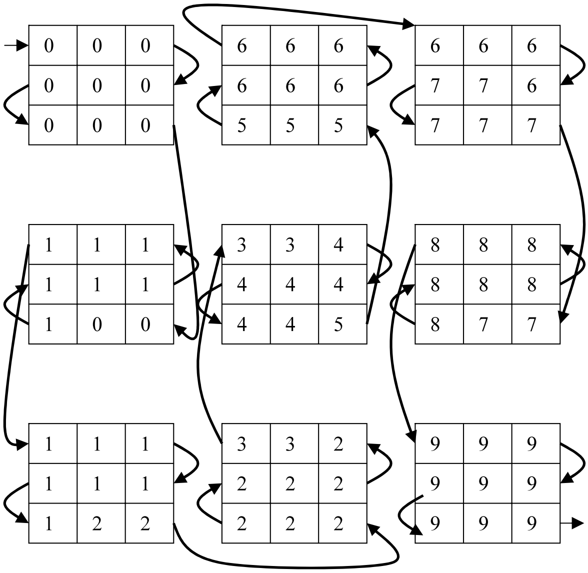


Fig. 38 Representation for a dry run of the proposed Algorithm

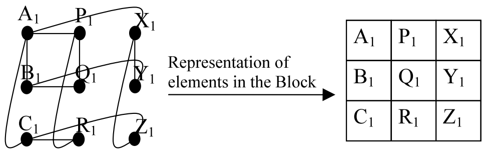


Fig. 39 Arbitrary input for Multi-Sort Algorithm (These data elements are used for a dry run of this algorithm to prove the correctness to the readers. Figures 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54 below are further operations performed on theses data elements. All Figs. 40–54 are connected to each other in the algorithmic flow)

0	5	9	4	4	6	1	1	2
4	2	8	8	6	2	0	1	1
9	1	7	9	9	7	0	0	8
7	1	7	8	0	0	1	0	9
5	0	1	1	2	3	3	3	6
6	1	8	1	3	6	6	7	8
4	2	0	9	4	4	5	5	0
0	9	4	2	2	6	8	6	5
9	2	1	6	6	1	9	7	1

of data after completion of above operations. Figures 38 and 39 provides representation for a dry run of the proposed Multi-Sort algorithm and its arbitrary inputs.

Stage 1

Step 1. $\forall \alpha, \beta, i, j : 1 \leq \alpha, \beta, i, j \leq n$ do in parallel
$B[\alpha, \beta, i, 1][1, j] \leftarrow A[\alpha, \beta, i, j]$

Step 2. $\forall \alpha, \beta, i : 1 \leq \alpha, \beta, i \leq n$ *AND* $j = 1$ do in parallel
$C[\alpha, i, \beta, n][1, j] \leftarrow B[\alpha, \beta, i, 1][1, j]$
Step 3. $\forall \alpha, \beta, i : 1 \leq \alpha, \beta, i \leq n$ do in parallel
Call Horiz_links($\alpha, \beta, i, n, n-1$)
$C[\alpha, \beta, i, k][1, j] \leftarrow C[\alpha, \beta, i, n][1, j]$
Step 4. $\forall \alpha, \beta, j : 1 \leq \alpha, \beta, j \leq n$ do in parallel
$C[\alpha, \beta, 1, j][i, 1] \leftarrow C[\alpha, \beta, i, j][1, j]$

Stage 2

Step 1. Call special_C_operation

Stage 3

Step 1. $\forall \alpha, \beta, j : 1 \leq \alpha, \beta, j \leq n$ do in parallel
$C[\alpha, \beta, i, j][1, j] \leftarrow C[\alpha, \beta, 1, j][i, 1]$
Step 2. $\forall \alpha, \beta, i : 1 \leq \alpha, \beta, i \leq n$ do in parallel
$C[\alpha, \beta, i, 1][1, j] \leftarrow C[\alpha, \beta, i, j][1, j]$
Step 3. $\forall \alpha, \beta, i : 1 \leq \alpha, \beta, i \leq n$ do in parallel
$C[\alpha, \beta, i, n][1, j] \leftarrow C[\alpha, \beta, i, 1][1, j]$
Step 4. $\forall \alpha, \beta, i : 1 \leq \alpha, \beta, i \leq n$ do in parallel
$C[\alpha, \beta, i, 1][1, j] \leftarrow C[\alpha, I, \beta, n,][1, j]$
Step 5. $\forall \alpha, \beta, i : 1 \leq \alpha, \beta, i \leq n$ do in parallel
$A[\alpha, \beta, i, j] \leftarrow C[\alpha, \beta, i, 1][1, j]$

2.4.1 Explanation (F operation)

All the stages of an F operation consist of steps and for each step the correctness parameters are explained using figures (which show the positions of data elements after implementation of respective steps).

Stage 1

Step 1. $\forall \alpha, \beta, i, j : 1 \leq \alpha, \beta, i, j \leq n$ do in parallel
$B[\alpha, \beta, i, 1][1, j] \leftarrow A[\alpha, \beta, i, j]$
Step 2. $\forall \alpha, \beta, i : 1 \leq \alpha, \beta, i \leq n$ *AND* $j = 1$ do in parallel
$C[\alpha, i, \beta, n][1, j] \leftarrow B[\alpha, \beta, i, 1][1, j]$
Step 3. $\forall \alpha, \beta, i : 1 \leq \alpha, \beta, i \leq n$ do in parallel
Call Horiz_links($\alpha, \beta, i, n, n-1$)
$C[\alpha, \beta, i, k][1, j] \leftarrow C[\alpha, \beta, i, n][1, j]$
Step 4. $\forall \alpha, \beta, j : 1 \leq \alpha, \beta, j \leq n$ do in parallel
$C[\alpha, \beta, 1, j][i, 1] \leftarrow C[\alpha, \beta, i, j][1, j]$

Stage 2

Step 1. Call special_C_operation

Stage 3

Step 1. $\forall \alpha, \beta, j : 1 \leq \alpha, \beta, j \leq n$ do in parallel
$C[\alpha, \beta, i, j][1, j] \leftarrow C[\alpha, \beta, 1, j][i, 1]$

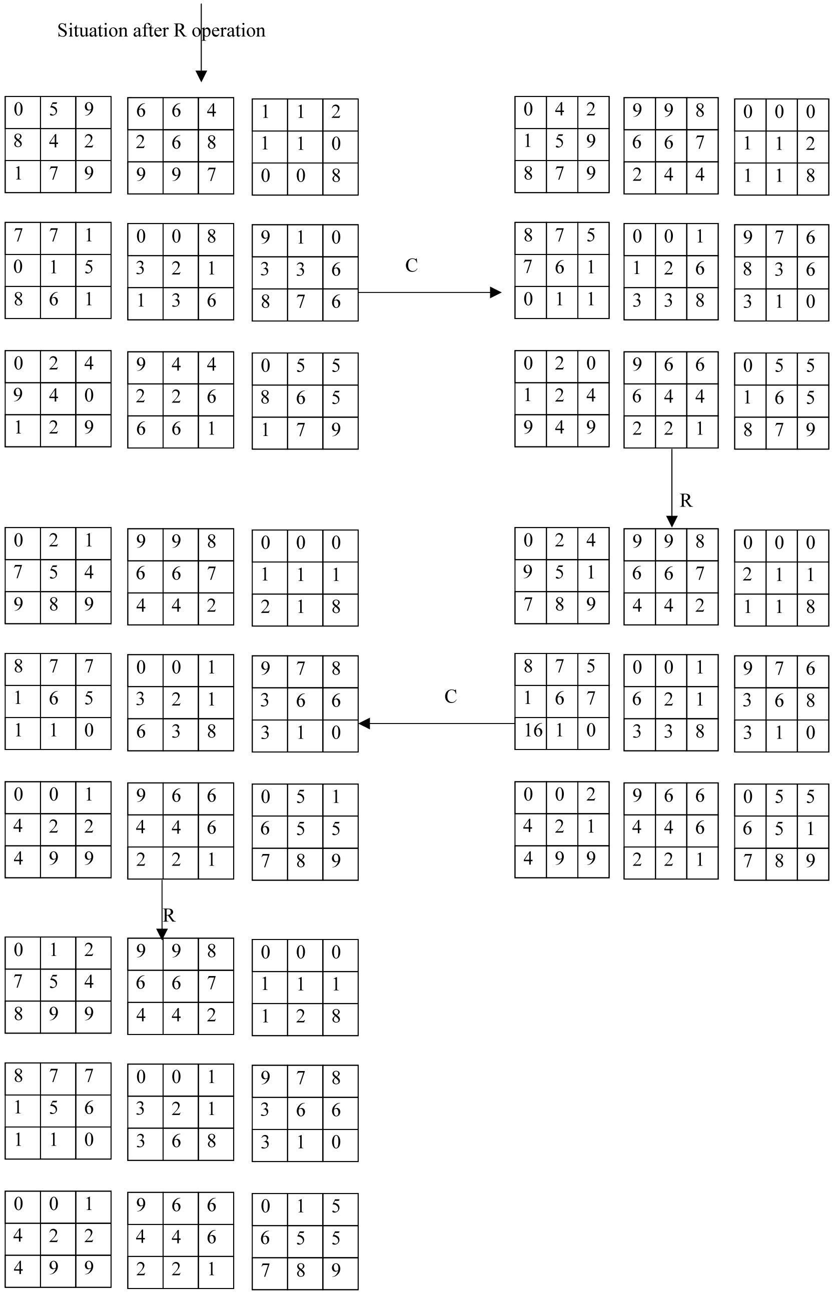


Fig. 40 The sorted blocks. Here Step 1 of the Algorithm is completed. The purpose of Step 1, i.e., the 2D Sort, is to sort the individual blocks in the snake-like row-major form. It can be easily seen that the aim has already been accomplished by performing $\log n$ of R–C operations. Moreover, an extra step of an R operation is also performed, as described by the algorithm. The 3D sort consists of three steps. We represent the third dimension Sort as T. The T operation itself is divided into three steps; we denote these operations as T_{S1}, T_{S2}, T_{S3}

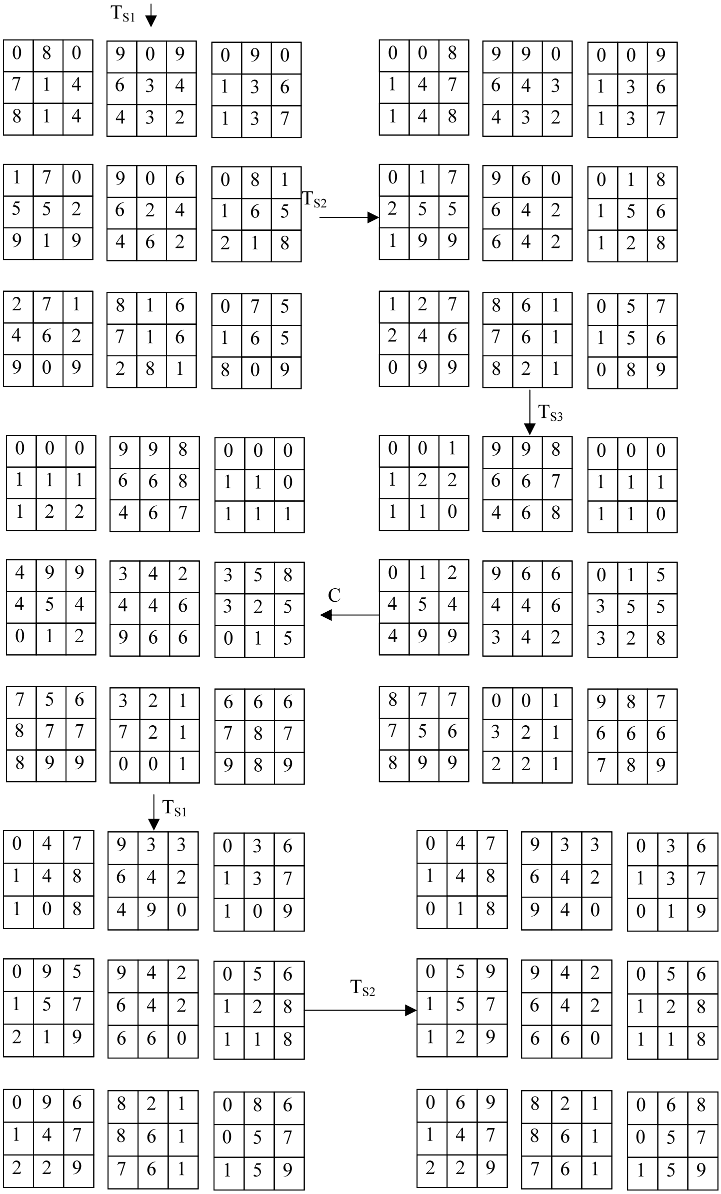


Fig. 41 3D Sort

Step 2. $\forall \alpha, \beta, i : 1 \leq \alpha, \beta, i \leq n$ do in parallel
$C[\alpha, \beta, i, 1][1, j] \leftarrow C[\alpha, \beta, i, j][1, j]$

Step 3. $\forall \alpha, \beta, i : 1 \leq \alpha, \beta, i \leq n$ do in parallel
$C[\alpha, \beta, i, n][1, j] \leftarrow C[\alpha, \beta, i, 1][1, j]$

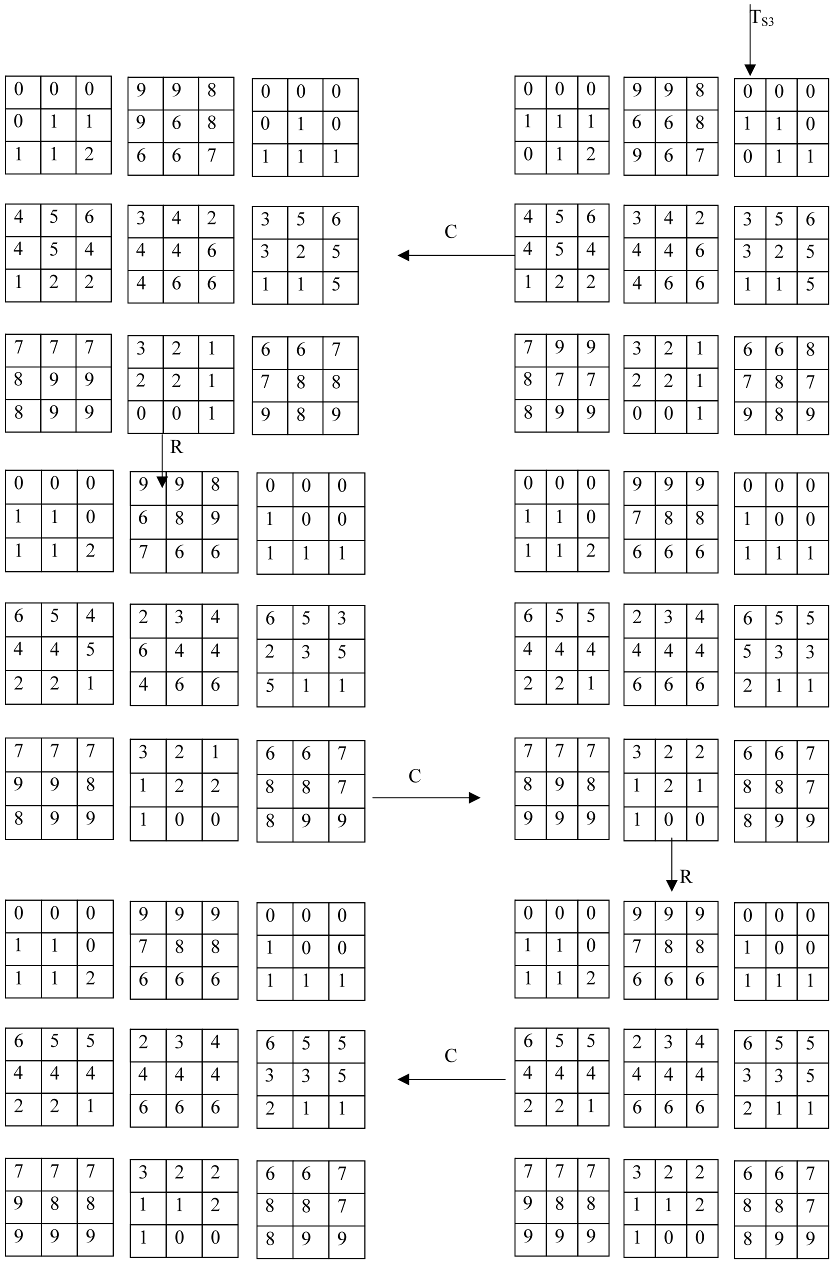


Fig. 42 3D Sort (continued)

Step 4. $\forall \alpha, \beta, i : 1 \leq \alpha, \beta, i \leq n$ do in parallel
$C[\alpha, \beta, i, 1][1, j] \leftarrow C[\alpha, I, \beta, n,][1, j]$

Step 5. $\forall \alpha, \beta, i : 1 \leq \alpha, \beta, i \leq n$ do in parallel
$A[\alpha, \beta, i, j] \leftarrow C[\alpha, \beta, i, 1][1, j]$

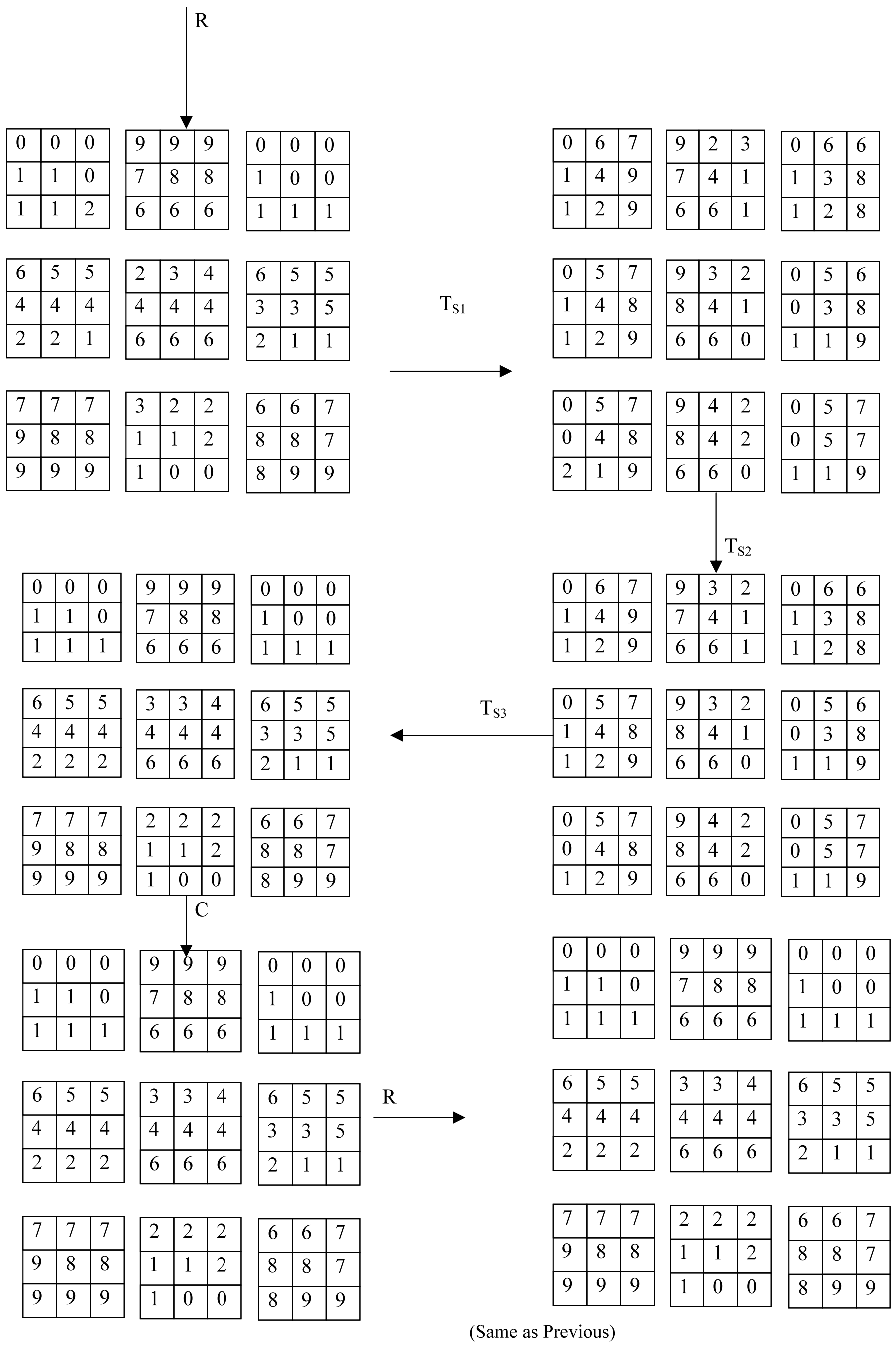


Fig. 43 3D Sort (continued)

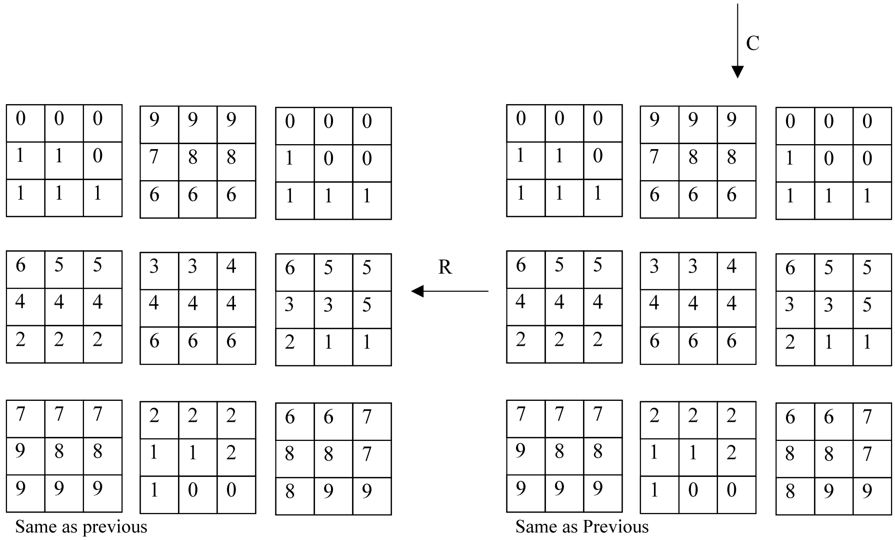


Fig. 44 3D Sort (continued)

2.5 Special C operation

This special C operation works the same as a C operation (as defined previously), the only difference is that now the direction of Sort depends upon the β value rather than the combined value of α and β.

Step 1. $\forall\alpha, \beta, i, j : 1 \le \alpha, \beta, i, j \le n$ do in parallel
$B[\alpha, \beta, 1, j][i, 1] \leftarrow A[\alpha, \beta, i, j]$
Step 2. $\forall\alpha, \beta : 1 \le \alpha, \beta \le n$ do in parallel
sort_ascend()
Step 3. $\forall\alpha, \beta, i, j : 1 \le \alpha, \beta, i, j \le n$ do in parallel
$A[\alpha, \beta, i, j] \leftarrow B[\alpha, \beta, 1, j][i, 1]$

2.6 Special R operation

The special R operation is applied on all blocks in the same direction.

Step 1. $\forall\alpha, \beta, i, j : 1 \le \alpha, \beta, i, j \le n$ do in parallel
$B[\alpha, \beta, 1, j][i, 1] \leftarrow A[\alpha, \beta, i, j]$
Step 2. $\forall\alpha, \beta : 1 \le \alpha, \beta \le n$
if (β *mod* $2 = 0$) then
$\forall\alpha, \beta : 1 \le \alpha, \beta \le n$ do in parallel
sort_ascend()
else
$\forall\alpha, \beta : 1 \le \alpha, \beta \le n$ do in parallel
sort_descend();
endif

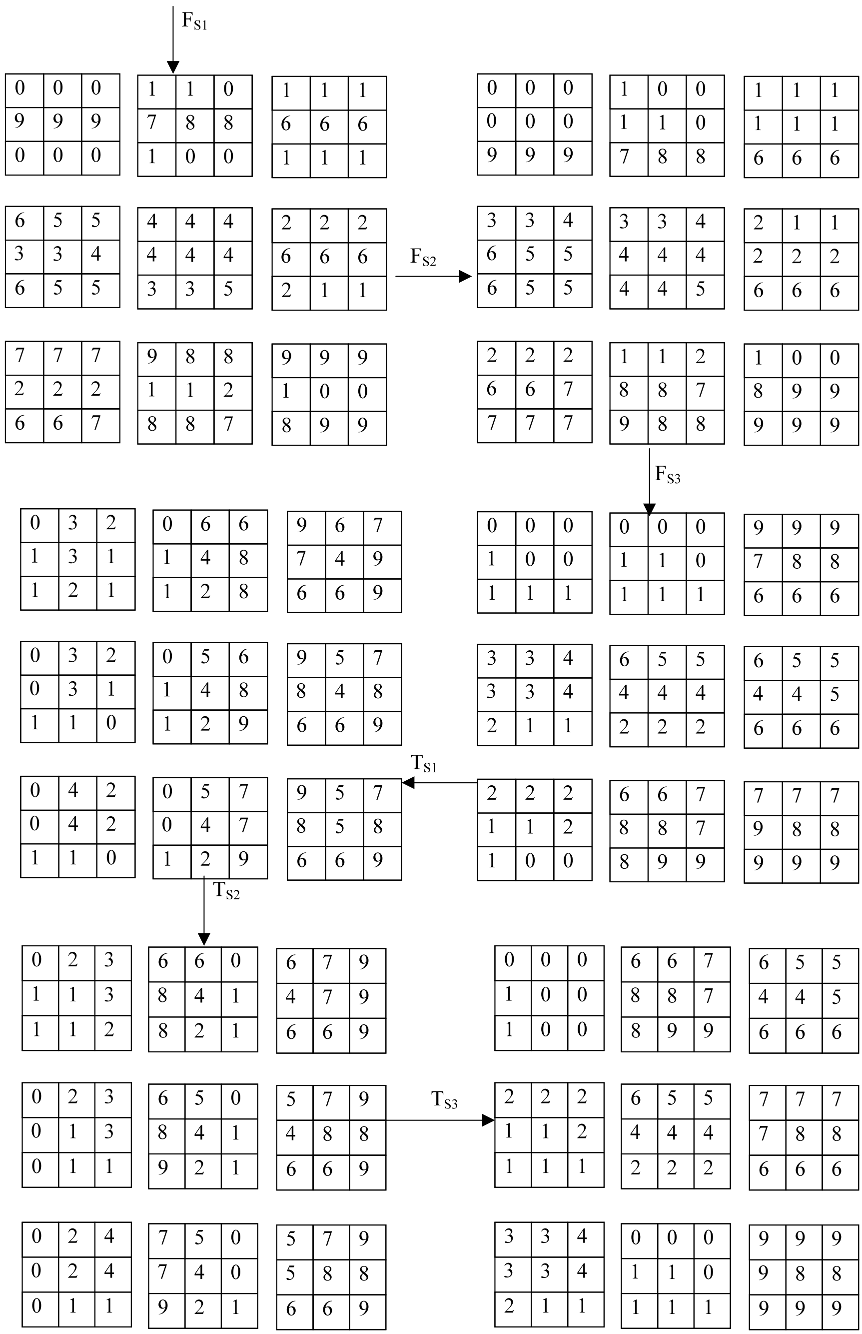


Fig. 45 2D Sort after 3D Sort (continued)

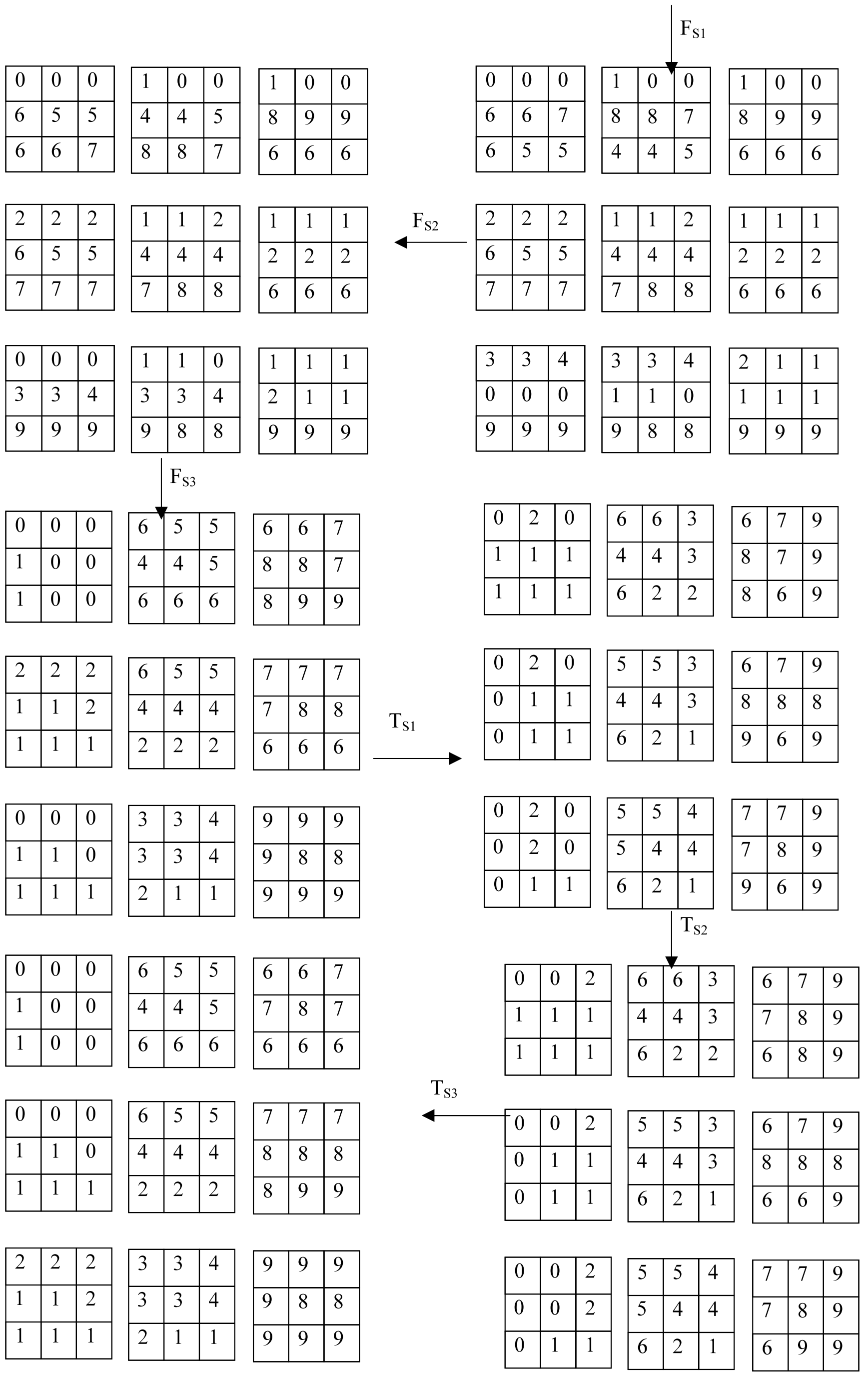


Fig. 46 2D Sort (continued)

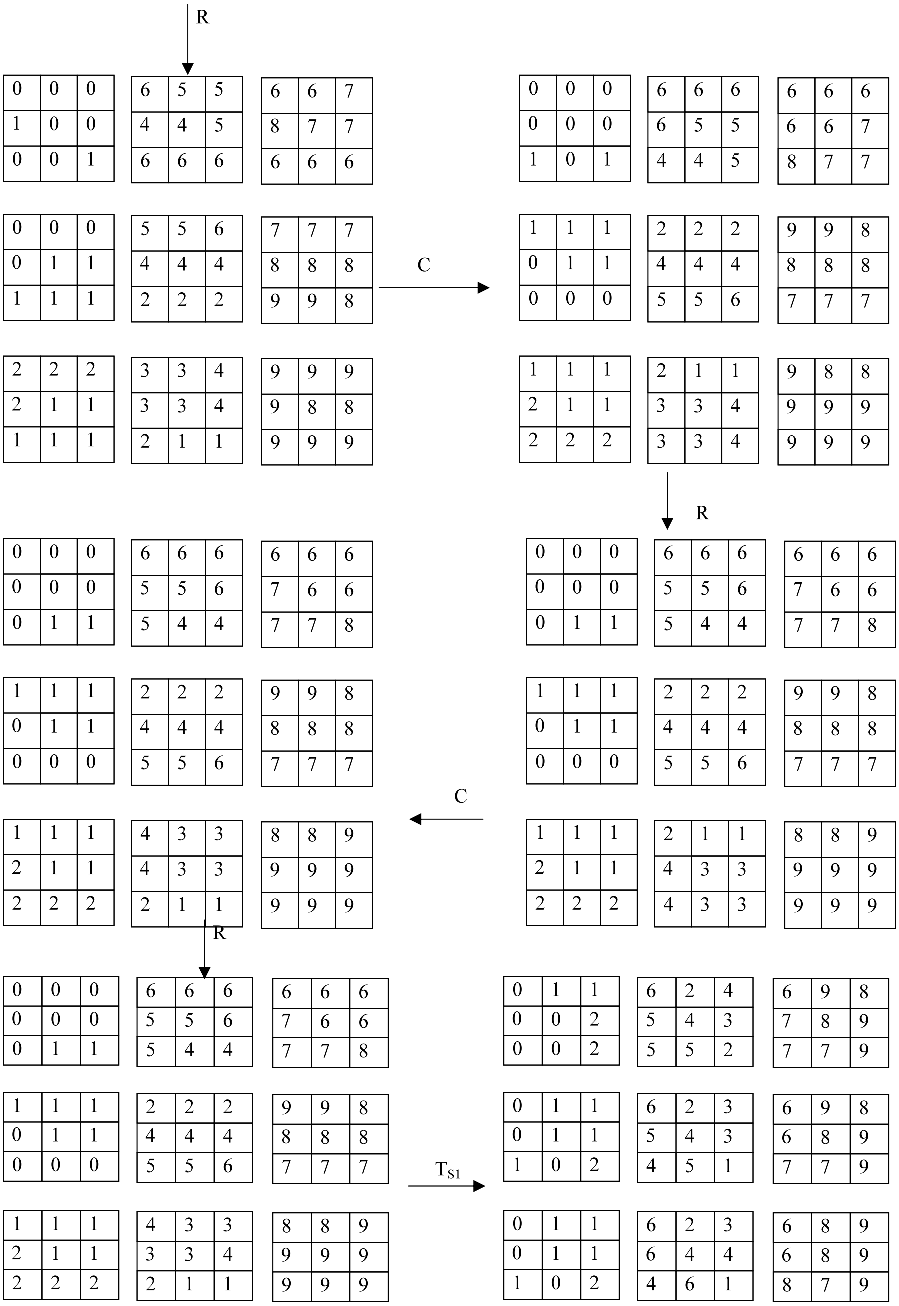


Fig. 47 2D Sort (continued)

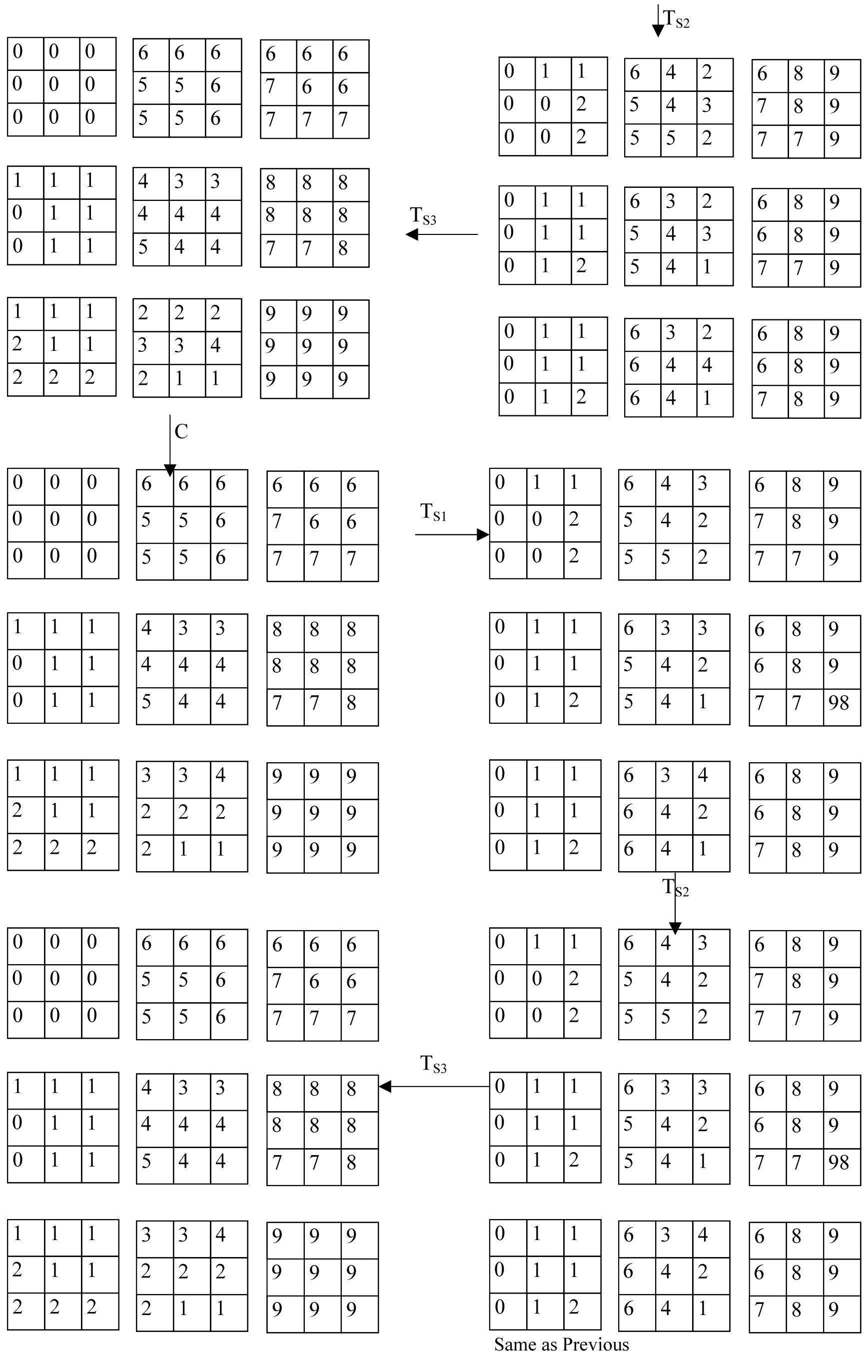


Fig. 48 2D Sort (continued)

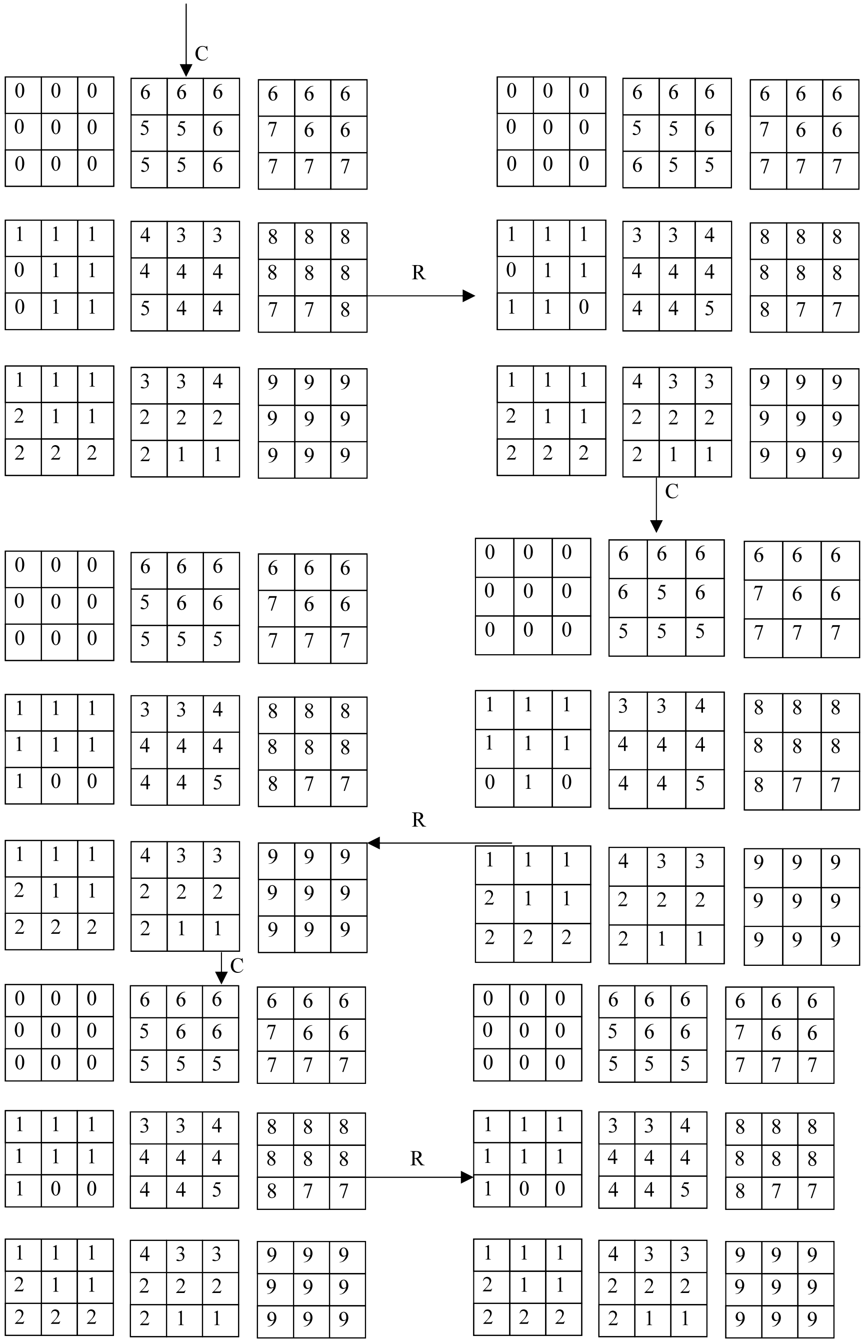


Fig. 49 2D Sort (continued)

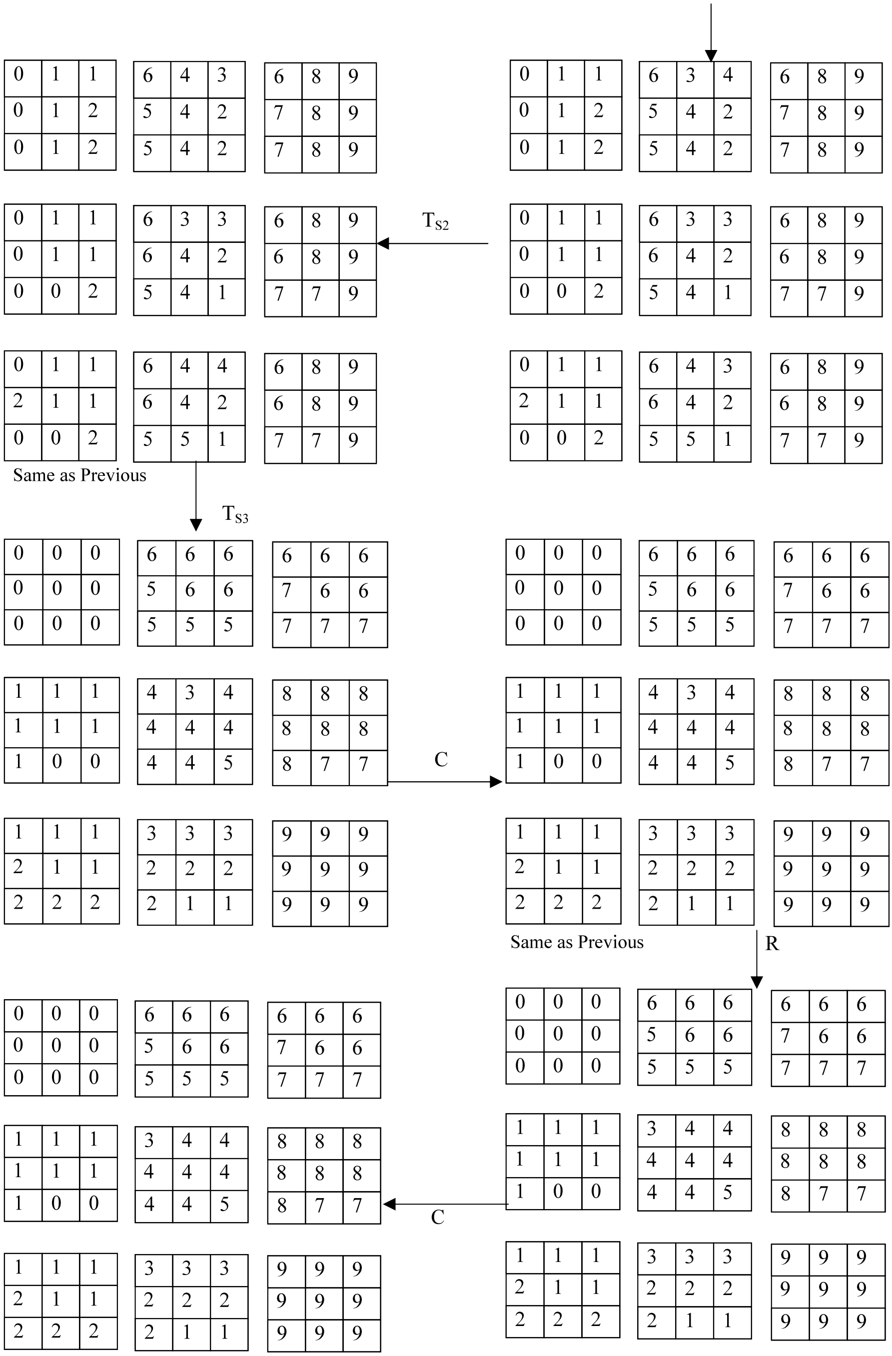


Fig. 50 2D Sort (continued)

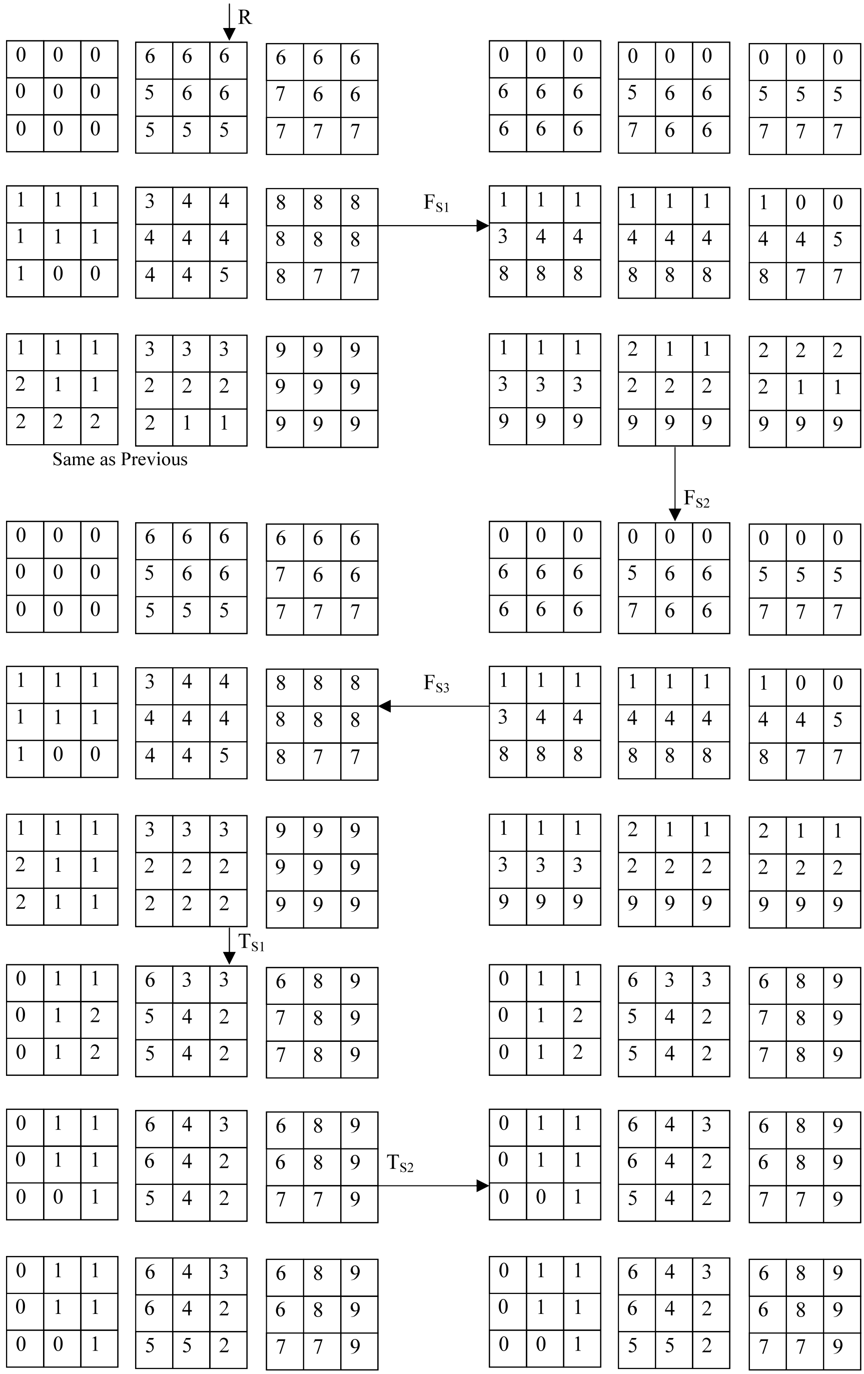


Fig. 51 2D Sort (continued)

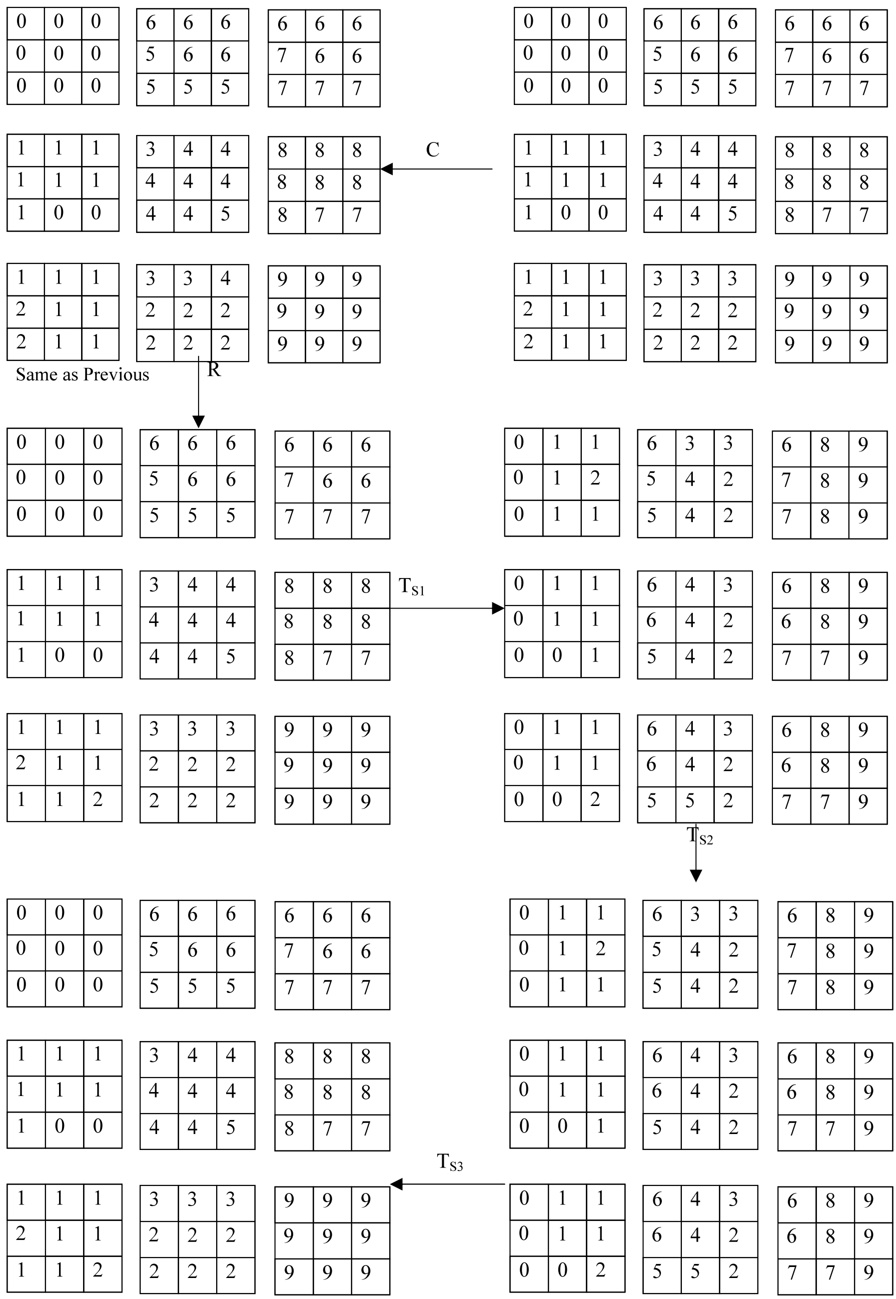


Fig. 52 2D Sort (continued)

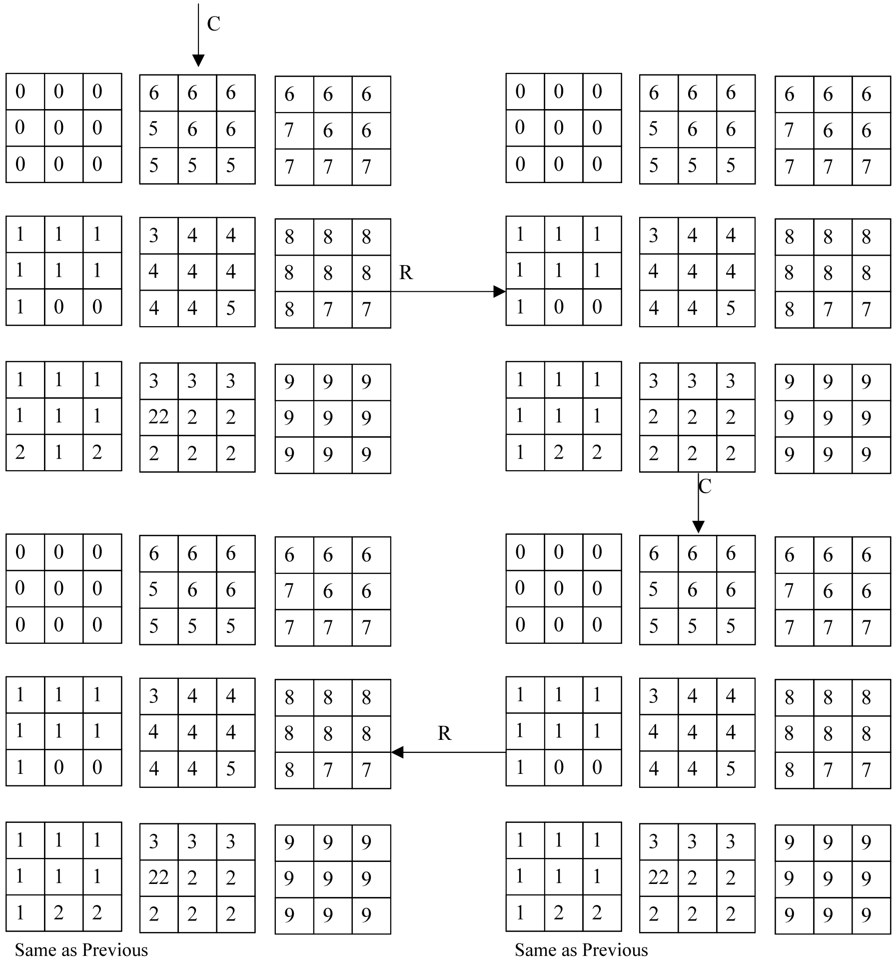


Fig. 53 2D Sort (continued)

Step 3. $\forall \alpha, \beta, i, j : 1 \leq \alpha, \beta, i, j \leq n$ do in parallel
$A[\alpha, \beta, i, j] \leftarrow B[\alpha, \beta, i, 1][1, j]$

3 Algorithm for sorting $N = n^4$ data elements

The algorithm for sorting data elements using MMT architecture consists of three steps: Step 1 which is 2D Sort, Step 2 which is 3D Sort, and Step 3 which is 4D 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 some optimal algorithm so that two consecutive blocks in any row or any column of processor blocks are sorted in reverse directions. To perform this step we do $\log n$ of R–C

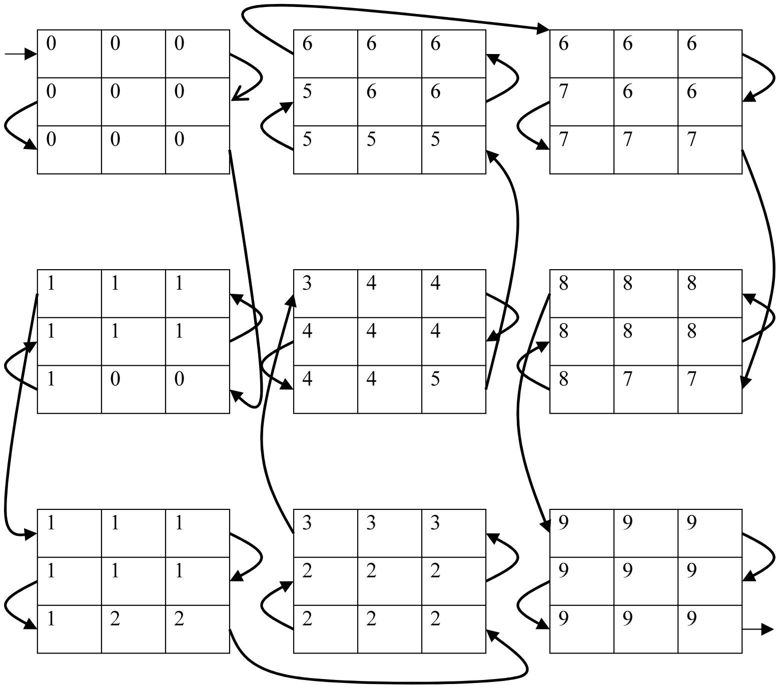


Fig. 54 Final output

operations, followed by an R operation. The output will be the individually sorted blocks.

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 $B(i, \beta)$, $1 \le i \le n$, are merged to produce one sorted 3D block, consisting of n^3 sorted elements, in the block-major snake-like ordering. For this merging, we 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.

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. The output after this step will be all sorted elements in the form given in Fig. 4.

4 Use of Algorithm

The purpose of a dry run of the proposed algorithm is shown in Fig. 55, and the dry run of the algorithm proposed above is shown in Fig. 56 for $N = 81$. The initial input

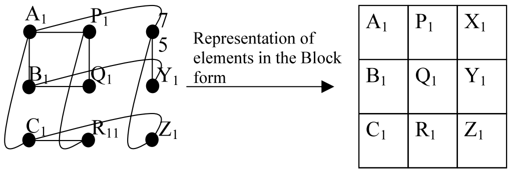


Fig. 55 Representation for a dry run of the proposed Algorithm

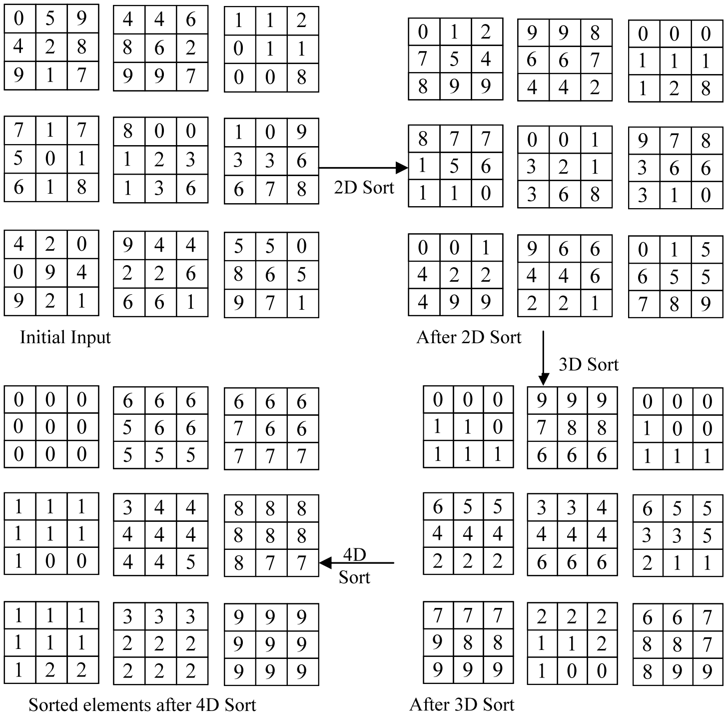


Fig. 56 The dry run of the algorithm with $N = 81$

is given to the 3 × 3 MMT (as given in Fig. 56) and 2D Sort is implemented to this data set. The purpose of Step 1, i.e., the 2D Sort, is to sort the individual blocks in the snake-like row-major form. It can be easily seen that the aim has already been achieved by performing $\log n$ of R–C operations. Moreover, an extra step of an R operation is also performed, as described by the algorithm.

In 3D Sort, after performing $\log n$ of T–C operations, the 2D sorting has been performed, i.e., the R–C–R–C–R operations, after that T–C–R–C–R is performed. After all the operations have been performed, the output is in a snake-like block-major form. (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. The output after this step will be all sorted elements in the form given in Fig. 4. The final dry run output of Multi-Sort algorithm on MMT is shown in Fig. 57.

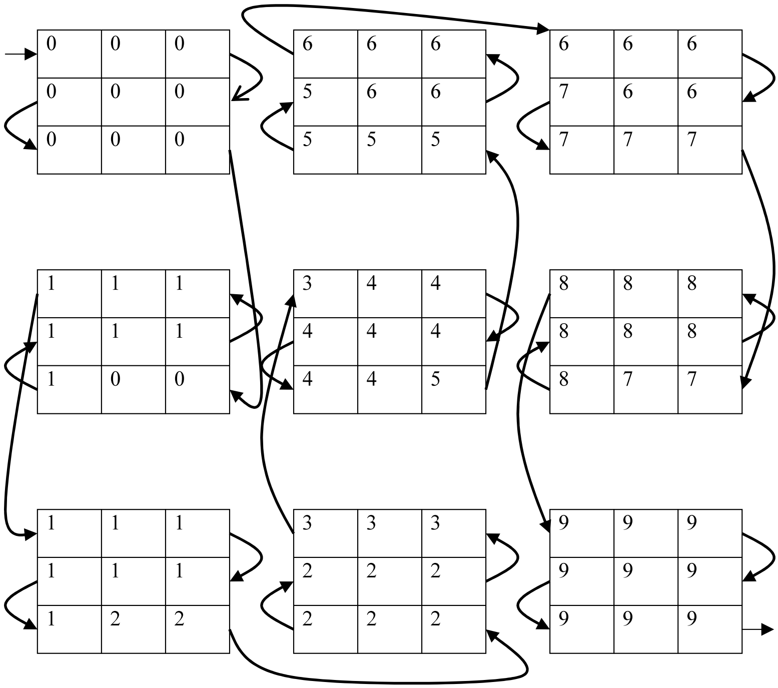


Fig. 57 Final output

5 Conclusion and future work

We have proposed a sorting algorithm, called Multi-Sort, on a recently developed architecture called Multi-Mesh of Trees (MMT). We have improved the time complexity of intrablock Sort. The time complexity of the compare-exchange step in MMT is same as that in MM, i.e., $O(n)$. The communication time complexity has been improved from $O(n)$ to $O(\log n)$. The communication time complexity for 2D Sort in MM is $O(n)$, whereas the same in MMT is $O(\log n)$. Additional time complexity of $O(n \log n)$ has been introduced for self-sort in order to generalize the algorithm for any number of elements.

The scope of complete time complexity can further be reduced for the compare–exchange step. This is possible if the algorithm is further analyzed based on the physical parameter of implementations with MMT architecture, and if more efficient parameters in the algorithm to conduct the 2D, 3D and 4D sorting are proposed.

Acknowledgements Very special thanks to Dr. Yajulu Medury, Chief Operating Officer, Jaypee Education System, for providing excellent Research Environment and Learning Ambience and Sun Solaris and High-end Computing Labs at JUIT.

References

1. Jana PK (2003) Multi-Mesh of Trees with its parallel algorithms. J Syst Archit, 193–206

2. De M, Das D, Ghosh M, Sinha BP (1997) An efficient sorting algorithm on multi-mesh. IEEE Trans Comput 46(10):1132–1137
3. Das D, Sinha BP (1995) Multi-mesh an efficient topology for parallel processing. In: Proceeding of the ninth international parallel processing symposium, 1995, pp 17–21
4. Das D, De M, Sinha BP (1999) A new network topology with multiple mesh. IEEE Trans Comput 48(5):536–551
5. Scherson ID, Sen S (1989) Parallel sorting algorithm in two-dimensional VLSI models of computation. IEEE Trans Comput 38(2):238–249
6. Preparata F (1978) New parallel sorting schemes. IEEE Trans Comput 27(7):669–673
7. Knuth DE (1973) The art of computer programming, sorting and searching, vol 3. Addison-Wesley, Reading
8. Akl SG, Santoro N (1987) Optimal parallel merging and sorting without memory conflicts. IEEE Trans Comput 36(11):1367–1369
9. Hwang K, Briggs FA (1989) Computer architecture and parallel processing. McGraw-Hill, New York
10. Arabnia HR, Oliver MA (1987) Arbitrary rotation of raster images with SIMD machine architectures. Int J Eurograph Assoc (Comput Graph Forum) 6(1):3–12
11. Bhandarkar SM, Arabnia HR, Smith JW (1995) A reconfigurable architecture for image processing and computer vision. Int J Pattern Recognit Artif Intell (IJPRAI) 9(2):201–229 (special issue on VLSI Algorithms and architectures for computer vision, image processing, pattern recognition and AI)
12. Bhandarkar SM, Arabnia HR (1995) The hough transform on a reconfigurable multi-ring network. J Parallel Distrib Comput 24(1):107–114
13. Arif Wani M, Arabnia HR (2003) Parallel edge-region-based segmentation algorithm targeted at reconfigurable multi-ring network. J Supercomput 25(1):43–63
14. Satish N, Harris M, Garland M (2009) Designing efficient sorting algorithms for manycore GPUs. In: IEEE international symposium on parallel & distributed processing, 2009, pp 1–10
15. Xie H, Xue Y (2008) An improved parallel sorting algorithm for odd sequence. In: International conference on advanced computer theory and engineering, 2008, pp 356–360
16. Nitin, Garhwal S, Srivastava N (2009) Designing a fault-tolerant fully-chained combining switches multi-stage interconnection network with disjoint paths. J Supercomput. doi:10.1007/s11227-009-0336-z
17. Akanda MdM, Abderazek BA, Sowa M (2008) Dual-execution mode processor architecture. J Supercomput 44(2):103–125