Basic Operations On The OTIS-Mesh Optoelectronic Computer*

Chih-fang Wang and Sartaj Sahni
Department of Computer and Information Science and Engineering
University of Florida
Gainesville, FL 32611
{wang,sahni}@cise.ufl.edu

Abstract

In this paper we develop algorithms for some basic operations – broadcast, window broadcast, prefix sum, data sum, rank, shift, data accumulation, consecutive sum, adjacent sum, concentrate, distribute, generalize, sorting, random access read and write – on the OTIS-Mesh [10] model. These operations are useful in the development of efficient algorithms for numerous applications [8].

1 Introduction

The Optical Transpose Interconnection System (OTIS), proposed by Marsden *et al.* [4], is a hybrid optical and electronic interconnection system for large parallel computers. The OTIS architecture employs free space optics to connect distant processors and electronic interconnect to connect nearby processors. Specifically, to maximize bandwidth, power efficiency, and to minimize system area and volume [1], the processors of an N^2 processor OTIS computer are partitioned into N groups of N processors each. Each processor is indexed by a tuple (G, P), $0 \leq G, P < N$, where G is the group index (*i.e.*, the group the processor is in), and P the processor index within a group. The inter group interconnects are optical while the intra group interconnects are electronic. The optical or OTIS interconnects connect pairs of processors of the form $[(G, P), (P, G)]$; that is, the group and processor indices are transposed by an optical interconnect. The electrical or intra group interconnections are according to any of the well studied electronic interconnection networks – mesh, hypercube, mesh of trees, and so forth. The choice if the electronic interconnection network defines a sub-family of OTIS computers – OTIS-Mesh, OTIS-Hypercube, and so forth. Figure 1 shows a 16 processor OTIS-Mesh. Each small square represents a processor. The number inside a processor square is the processor index P. Some processor squares have a pair (P_x, P_y) inside them. The pair gives the row and column index of the processor P within its $\sqrt{N} \times \sqrt{N}$ mesh. Each large

*This work was supported, in part, by the Army Research Office under grant DAA H04-95-1-0111.

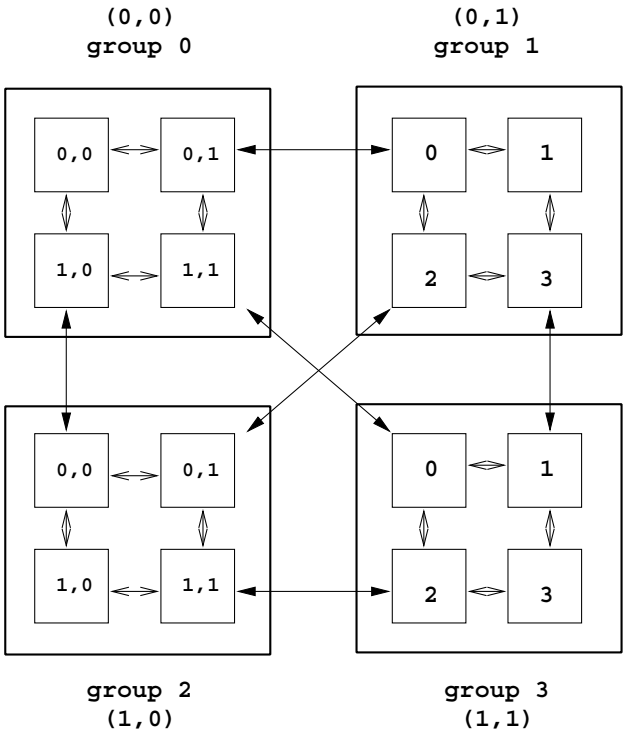

Figure 1: 16 processor OTIS-Mesh

square encloses a group of processors. A group index G may also be given as a pair (G_x, G_y) where G_x and G_y are the row and column indices of the group assuming a $\sqrt{N} \times \sqrt{N}$ layout of groups.

Zane *et al.* [11] have shown that an N^2 processor OTIS-Mesh can simulate each move of a $\sqrt{N} \times \sqrt{N} \times \sqrt{N} \times \sqrt{N}$ four-dimensional (4D) mesh computer using either one electronic move or one electronic and two OTIS moves (depending on which dimension of the 4D mesh we are to move along). They have also shown that an N^2 processor OTIS-Hypercube can simulate each move of an N^2 processor hypercube using either one electronic move or one electronic and two OTIS moves. Sahni and Wang [10, 9] have developed efficient algorithms to rearrange data according to bit-permute-complement (BPC) [5] permutations on OTIS-Mesh and OTIS-Hypercube computers, respectively. Rajasekaran and Sahni [7] have developed efficient randomized algorithms for routing, selection, and sorting on an OTIS-Mesh.

In this paper, we develop deterministic OTIS-Mesh algorithms for the basic data operations for parallel computation that are studied in [8]. As shown in [8], algorithms for these operations can be used to arrive at efficient parallel algorithms for numerous applications, from image processing, computational geometry, matrix algebra, graph theory, and so forth.

We consider both the synchronous SIMD and synchronous MIMD models. In both, all pro-

cessors operate in lock-step fashion. In the SIMD model, all active processors perform the same operation in any step and all active processors move data along the same dimension or along OTIS connections. In the MIMD model, processors can perform different operations in the same step and can move data along different dimensions.

2 Basic Operations

2.1 Data Broadcast

Data broadcast is, perhaps, the most fundamental operation for a parallel computer. In this operation, data that is initially in a single processor (G, P) is to be broadcast or transmitted to all N^2 processors of the OTIS-Mesh. Data broadcast can be accomplished using the following three step algorithm:

Step 1: Processor (G, P) broadcasts its data to all other processors in group G.

Step 2: Perform an OTIS move.

Step 3: Processor G of each group broadcasts the data within its group.

Following Step 2, one processor of each group has a copy of the data, and following Step 3 each processor of the OTIS-Mesh has a copy. In the SIMD model, Steps 1 and 3 take $2(\sqrt{N}-1)$ electronic moves each, and Step 2 takes one OTIS move. The SIMD complexity is $4(\sqrt{N}-1)$ electronic moves and 1 OTIS move, or a total of $4\sqrt{N}-3$ moves. Note that our algorithm is optimal because the diameter of the OTIS-Mesh is $4\sqrt{N}-3$ [10]. For example, if the data to be broadcast is initially in processor (0,0), the data needs to reach processor $(N-1, N-1)$, which is at a distance of $4\sqrt{N}-3$. In the MIMD model, the complexity of Steps 1 and 3 depends on the value of $P = (P_x, P_y)$ and ranges from a low of approximately $\sqrt{N}-1$ to a high of $2(\sqrt{N}-1)$. The overall complexity is at most $4(\sqrt{N}-1)$ electronic moves and one OTIS move. By contrast, simulating the 4D-mesh broadcast algorithm using the simulation method of [11] takes $4(\sqrt{N}-1)$ electronic moves and $4(\sqrt{N}-1)$ OTIS moves in the SIMD model and up to this many moves in the MIMD model.

2.2 Window Broadcast

In a window broadcast, we start with data in the top left $w \times w$ submesh of a single group G. Here w divides $\sqrt{N}$. Following the window broadcast operation, the initial $w \times w$ window tiles all

groups; that is, the window is broadcast both within and across groups. Our algorithm for window broadcast is:

Step 1: Do a window broadcast within group G.

Step 2: Perform an OTIS move.

Step 3: Do an intra group data broadcast from processor G of each group.

Step 4: Perform an OTIS move.

Following Step 1 the initial window properly tiles group G and we are left with the task of broadcasting from group G to all other groups. In Step 2, data $d(G,P)$ from (G,P) is moved to (P,G) for $0 \leq P < N$. In Step 3, $d(G,P)$ is broadcast to all processors (P,i), $0 \leq P,i < N$, and in Step 4 $d(G,P)$ is moved to (i,P), $0 \leq i,P < N$.

Step 1 of our window broadcast algorithm takes $2(\sqrt{N}-w)$ electronic moves in both the SIMD and MIMD models, and Step 3 takes $2(\sqrt{N}-1)$ electronic moves in the SIMD model and up to $2(\sqrt{N}-1)$ electronic moves in the MIMD model. The total cost is $4\sqrt{N}-2w-2$ electronic and 2 OTIS moves in the SIMD model and up to this many moves in the MIMD model. A simulation of the 4D mesh window broadcast algorithm takes the same number of electronic moves, but also takes $4(\sqrt{N}-1)$ OTIS moves.

2.3 Prefix Sum

The index (G,P) of a processor may be transformed into a scalar $I = GN+P$ with $0 \leq I < N^2$. Let $D(I)$ be the data in processor I, $0 \leq I < N^2$. In a prefix sum, each processor I computes $S(I) = \sum_{i=0}^{I} D(i)$, $0 \leq I < N^2$. A simple prefix sum algorithm results from the following observation:

$$S(I) = SD(I) + LP(I)$$

where $SD(I)$ is the sum of $D(i)$ over all processors i that are in a group smaller than the group of I and $LP(I)$ is the local prefix sum within the group of I. The simple prefix sum algorithm is:

Step 1: Perform a local prefix sum in each group.

Step 2: Perform an OTIS move of the prefix sums computed in Step 1 for all processors $(G,N-1)$.

Step 3: Group $N-1$ computes a modified prefix sum of the values, A, received in Step 2. In this modification, processor P computes $\sum_{i=0}^{P-1} A(i)$ rather than $\sum_{i=0}^{P} A(i)$.

Step 4: Perform an OTIS move of the modified prefix sums computed in Step 3.

Step 5: Each group does a local broadcast of the modified prefix sum received by its $N-1$ processor.

Step 6: Each processor adds the local prefix sum computed in Step 1 and the modified prefix sum it received in Step 5.

The local prefix sums of Steps 1 and 3 take $3(\sqrt{N}-1)$ electronic moves in both the SIMD and MIMD models, and the local data broadcast of Step 5 takes $2(\sqrt{N}-1)$ electronic moves. The overall complexity is $8(\sqrt{N}-1)$ electronic moves and 2 OTIS moves. This can be reduced to $7(\sqrt{N}-1)$ electronic moves and 2 OTIS moves by deferring some of the Step 1 moves to Step 5 as below.

Step 1: In each group, compute the row prefix sums R.

Step 2: Column $\sqrt{N}-1$ of each group computes the modified prefix sums of its R values.

Step 3: Perform an OTIS move on the prefix sums computed in Step 2 for all processors $(G, N-1)$.

Step 4: Group $N-1$ computes a modified prefix sum of the values, A, received in Step 3.

Step 5: Perform an OTIS move of the modified prefix sums computed in Step 4.

Step 6: Each group broadcasts the modified prefix sum received in Step 5 along column $\sqrt{N}-1$ of its mesh.

Step 7: The column $\sqrt{N}-1$ processors add the modified prefix sum received in Step 6 and the prefix sum of R values computed in Step 2 minus its own R value computed in Step 1.

Step 8: The result computed by column $\sqrt{N}-1$ processors in Step 7 is broadcast along mesh rows.

Step 9: Each processor adds its R value and the value it received in Step 8.

If we simulate the best 4D mesh prefix sum algorithm, the resulting OTIS mesh algorithm takes $7(\sqrt{N}-1)$ electronic and $6(\sqrt{N}-1)$ OTIS moves.

2.4 Data Sum

In this operation, each processor is to compute the sum of the D values of all processors. An optimal SIMD data sum algorithm is:

Step 1: Each group performs the data sum.

Step 2: Perform an OTIS move.

Step 3: Each group performs the data sum.

In the SIMD model Steps 1 and 3 take $4(\sqrt{N}-1)$ electronic moves, and step 2 takes 1 OTIS move. The total cost is $8(\sqrt{N}-1)$ electronic and 1 OTIS moves. Note that since the distance between processors $(0,0)$ and $(N-1,N-1)$ is $4(\sqrt{N}-1)$ electronic and 1 OTIS moves and since each needs to get information from the other, at least $8(\sqrt{N}-1)$ electronic and 1 OTIS moves are needed (the moves needed to send information from $(0,0)$ to $(N-1,N-1)$ and those from $(N-1,N-1)$ to $(0,0)$ cannot be overlapped in the SIMD model). Also, note that a simulation of the 4D mesh data sum algorithm takes $8(\sqrt{N}-1)$ electronic and $8(\sqrt{N}-1)$ OTIS moves.

The MIMD complexity can be reduced by computing the group sums in the middle processor of each group rather than in the bottom right processor. The complexity now becomes $4(\sqrt{N}-1)$ electronic and 1 OTIS moves when $\sqrt{N}$ is odd and $4\sqrt{N}$ electronic and 1 OTIS moves when $\sqrt{N}$ is even. The simulation of the 4D mesh, however, takes $4(\sqrt{N}-1)$ electronic and $4(\sqrt{N}-1)$ OTIS moves. Notice that the MIMD algorithm is near optimal as the diameter of the OTIS-Mesh is $4\sqrt{N}-3$ [10].

2.5 Rank

In the rank operation, each processor I has a flag $S(I) \in \{0,1\}$, $0 \leq I < N^2$. We are to compute the prefix sums of the processors with $S(I)=1$. This operation can be performed in $7(\sqrt{N}-1)$ electronic and 2 OTIS moves using the prefix sum algorithm of Section 2.3.

2.6 Shift

Although there are many variations of the shift operation, the ones we believe are most useful in application development are:

(a) mesh row shift with zero fill — in this we shift data from processor (G_x, G_y, P_x, P_y) to processor (G_x, G_y, P_x, P_y+s), $-\sqrt{N} < s < \sqrt{N}$. The shift is done with zero fill and end discard (*i.e.*,

if $P_y + s > \sqrt{N}$ or $P_y + s < 0$, the data from P_y is discarded).

(b) mesh column shift with zero fill — similar to (a), but along mesh column P_x.

(c) circular shift on a mesh row — in this we shift data from processor (G_x, G_y, P_x, P_y) to processor $(G_x, G_y, P_x, (P_y + s) \bmod \sqrt{N})$.

(d) circular shift on a mesh column — similar to (c), but instead P_x is used.

(e) group row shift with zero fill — similar to (a), except that G_y is used in place of P_y.

(f) group column shift with zero fill — similar to (e), but along group column G_x.

(g) circular shift on a group row — similar to (c), but with G_y rather than P_y.

(h) circular shift on a group column — similar to (g), with G_x in place of G_y.

Shifts of types (a) through (d) are done using the best mesh algorithms while those of types (e) through (h) are done as below:

Step 1: Perform an OTIS move.

Step 2: Do the shift as a P_x (if originally a G_x shift) or a P_y (if originally a G_y shift) shift.

Step 3: Perform an OTIS move.

Shifts of types (a) and (b) take s electronic moves on the SIMD and MIMD models; (c) and (d) take $\sqrt{N}$ electronic moves on the SIMD model and $\max\{|s|, \sqrt{N} - |s|\}$ electronic moves on the MIMD model; (e) and (f) take s electronic and 2 OTIS moves on both SIMD and MIMD models; and (g) and (h) take $\sqrt{N}$ electronic and 2 OTIS moves on the SIMD model and $\max\{|s|, \sqrt{N} - |s|\}$ electronic and 2 OTIS moves on the MIMD model.

If we simulate the corresponding 4D mesh algorithms, we obtain the same complexity for (a) – (d), but (e) and (f) take an additional $2s - 2$ OTIS moves, and (g) and (h) take an additional $2 \times \max\{|s|, \sqrt{N} - |s|\} - 2$ OTIS moves.

2.7 Data Accumulation

Each processor is to accumulate M, $0 \leq M \leq \sqrt{N}$, values from its neighboring processors along one of the four dimensions G_x, G_y, P_x, P_y. Let $D(G_x, G_y, P_x, P_y)$ be the data in processor

(G_x, G_y, P_x, P_y). In a data accumulation along the G_x dimension (for example), each processor (G_x, G_y, P_x, P_y) accumulates in an array A the data values from $((G_x + i) \bmod \sqrt{N}, G_y, P_x, P_y)$, $0 \leq i < M$. Specifically, we have

$$A[i] = D((G_x + i) \bmod \sqrt{N}, G_y, P_x, P_y)$$

Accumulation in other dimensions is similar.

The accumulation operation can be done using a circular shift of $-M$ in the appropriate dimension. The complexity is readily obtained from that for the circular shift operation (see Section 2.6).

2.8 Consecutive Sum

The N^2 processor OTIS-Mesh is tiled with one-dimensional blocks of size M. These blocks may align with any of the four dimensions G_x, G_y, P_x, and P_y. Each processor has M values $X[j]$, $0 \leq j < M$. The ith processor in a block is to compute the sum of the $X[i]$s in that block. Specifically, processor i of a block computes

$$S(i) = \sum_{j=0}^{M-1} X[i](j), 0 \leq i < M$$

where i and j are indices relative to a block.

When the one-dimensional blocks of size M align with the P_x or P_y dimensions, a consecutive sum can be performed by using M tokens in each block to accumulate the M sums $S(i)$, $0 \leq i < M$. Assume the blocks align along P_x. Each processor in a block initiates a token labeled with the processor's intra block index. The tokens from processors 0 through $M-2$ are right bound and that from $M-1$ is left bound. In odd time steps, right bound tokens move one processor right along the block, and in even time steps left bound tokens move one processor left along the block. When a token reaches the rightmost or leftmost processor in the block, it reverses direction. Each token visits each processor in its block twice – once while moving left and once while moving right. During the rightward visits it adds in the appropriate X value from the processor. After $4(M-1)$ time steps (and hence $4(M-1)$ electronic moves), all tokens return to their originating processors, and we are done.

In the MIMD model, the left and right moves can be done simultaneously, and only $2(M-1)$ electronic moves are needed.

When the one-dimensional size M blocks align with G_x or G_y, we first do an OTIS move; then run either a P_x or P_y consecutive sum algorithm; and then do an OTIS move. The number of electronic moves is the same as for P_x or P_y alignment. However, two additional OTIS moves are needed.

Simulation of the corresponding 4D mesh algorithm takes an additional $8M - 10$ OTIS moves for the case of G_x or G_y alignment in the SIMD model and an additional $4M - 6$ OTIS moves in the MIMD model.

2.9 Adjacent Sum

This operation is similar to the data accumulation operation of Section 2.7 except that the M accumulated values are to be summed. The operation can be done with the same complexity as data accumulation using a similar algorithm.

2.10 Concentrate

A subset of the processors contain data. These processors have been ranked as in Section 2.5. So the data is really a pair (D, r); D is the data in the processor and r is its rank. Each pair (D, r) is to be moved to processor r, $0 \leq r < b$, where b is the number of processors with data. Using the (G, P) format for a processor index, we see that (D, r) is to be routed from its originating processor to processor $(\lfloor r/N \rfloor, r \bmod N)$. We accomplish this using the steps:

Step 1: Each pair (D, r) is routed to processor $r \bmod N$ within its current group.

Step 2: Perform an OTIS move.

Step 3: Each pair (D, r) is routed to processor $\lfloor r/N \rfloor$ within its current group.

Step 4: Perform an OTIS move.

Theorem 1 *The four step algorithm given above correctly routes every pair (D, r) to processor $(\lfloor r/N \rfloor, r \bmod N)$.*

Proof Step 1 does the routing on the second coordinate. This step does not route two pairs to the same processor provided no group has two pairs (D_1, r_1), (D_2, r_2) with $r_1 \bmod N = r_2 \bmod N$. Since each group has at most N pairs and the ranks of these pairs are contiguous integers, no group can have two pairs with $r_1 \bmod N = r_2 \bmod N$. So following Step 1 each processor has at most one pair and each pair is in the correct processor of the group, though possibly in the wrong group.

To get the pairs to their correct groups without changing the within group index, Step 2 performs an OTIS move, which moves data from processor (G, P) to processor (P, G). Now all pairs in a group have the same r mod N value and different $\lfloor r/N \rfloor$ values. The routing on the $\lfloor r/N \rfloor$ values, as in Step 3, routes at most one pair to each processor. The OTIS move of Step 4, therefore, gets every pair to its correct destination processor. □

In group 0, Step 1 is a concentrate localized to the group, and in the remaining groups, Step 1 is a generalized concentrate in which the ranks have been increased by the same amount. In all groups we may use the mesh concentrate algorithm of [6] to accomplish the routing in $4(\sqrt{N} - 1)$ electronic moves. Step 3 is also a concentrate as the $\lfloor r/N \rfloor$ values of the pairs are in ascending order from $0, 1, 2, \cdots$. So Steps 1 and 3 take $4(\sqrt{N} - 1)$ electronic moves each in the SIMD model and $2(\sqrt{N} - 1)$ in the MIMD model [6]. Therefore, the overall complexity of concentrate is $8(\sqrt{N} - 1)$ electronic and 2 OTIS moves in the SIMD model and $4(\sqrt{N} - 1)$ electronic and 2 OTIS moves in the MIMD model.

We can improve the SIMD time to $7(\sqrt{N} - 1)$ electronic and 2 OTIS moves by using a better mesh concentrate algorithm than the one in [6]. The new and simpler algorithm is given below for the case of a generalized concentration on a $\sqrt{N} \times \sqrt{N}$ mesh.

Step 1: Move data that is to be in a column right of the current one rightwards to the proper processor in the same row.

Step 2: Move data that is to be in a column left of the current one leftwards to the proper processor in the same row.

Step 3: Move data that is to be in a smaller row upwards to the proper processor in the same column.

Step 4: Move data that is to be in a bigger row downwards to the proper processor in the same column.

In a concentrate operation on a square mesh data that begins in two processors of the same row ends up in different columns as the rank of these two data differs by at most $\sqrt{N} - 1$. So Steps 1 and 2 do not leave two or more data in the same processor. Steps 3 and 4 get data to the proper row and hence to the proper processor. Note that it is possible to have up to two data items in a processor following Step 1 and Step 3. The complexity of the above concentrate algorithm is

$4(\sqrt{N}-1)$ on a SIMD mesh and $2(\sqrt{N}-1)$ on an MIMD mesh (we can overlap Steps 1 and 2 as well as Steps 3 and 4 on an MIMD mesh).

For an ordinary concentrate in which the ranks begin at 1, Step 4 can be omitted as no data moves down a column to a row with bigger index. So an ordinary concentrate takes only $3(\sqrt{N}-1)$ moves. This improves the SIMD concentration algorithm of [6], which takes $4(\sqrt{N}-1)$ moves to do an ordinary concentrate.

Actually, we can show that the four step concentration algorithm just stated is optimal for the SIMD model. Consider the ordinary concentrate instance in which the selected elements are in processors $(0, \sqrt{N}-1)$, $(1, \sqrt{N}-2)$, $\cdots$, $(\sqrt{N}-1, 0)$. The ranks are 0, 1, $\cdots$, $\sqrt{N}-1$. So the data in processor $(0, \sqrt{N}-1)$ is to be moved to processor (0,0). This requires moves that yield a net of $\sqrt{N}-1$ left moves. Also, the data in processor $(\sqrt{N}-1, 0)$ is to be moved to processor $(0, \sqrt{N}-1)$. This requires a net of $\sqrt{N}-1$ upward moves and $\sqrt{N}-1$ rightward moves. None of these moves can be overlapped in the SIMD model. So every SIMD concentrate algorithm must take at least $\sqrt{N}-1$ moves in each of the directions left, right, and up; a total of at least $3(\sqrt{N}-1)$ moves.

For the generalized concentrate algorithm, the ranks need not start at zero. Suppose we have two elements to concentrate. One is at processor (0,0) and has rank $N-1$, and the other is at processor $(\sqrt{N}-1, \sqrt{N}-1)$ and has rank N. The data in (0,0) is to be moved to $(\sqrt{N}-1, \sqrt{N}-1)$ at a cost of $\sqrt{N}-1$ net right and down moves. The data in $(\sqrt{N}-1, \sqrt{N}-1)$ is to be moved to (0,0) at a cost of $\sqrt{N}-1$ net left and up moves. So at least $4(\sqrt{N}-1)$ moves are needed.

Theorem 2 *The OTIS-Mesh data concentration algorithm described above is optimal for both the SIMD and MIMD models; that is,* (a) *every SIMD concentration algorithm must make* $7(\sqrt{N}-1)$ *electronic and 2 OTIS moves in the worst case, and* (b) *every MIMD concentration algorithm must make* $4(\sqrt{N}-1)$ *electronic and 2 OTIS moves.*

Proof *(a)* Suppose that the data to be concentrated are in the processors shown in Table 1. Let a denote processor $(\sqrt{N}-1, \sqrt{N}-1, \sqrt{N}-1, \sqrt{N}-1)$, let b denote processor $(\sqrt{N}-1, 0, \sqrt{N}-1, 0)$, and let c denote processor (0,1,0,0). The ranks of a, b, and c are $N^{3/2}$, $N^{3/2}-N+\sqrt{N}-1$, and $N-\sqrt{N}$ respectively. Therefore, following the concentration the data $D(a)$, $D(b)$, and $D(c)$ initially in processors a, b, and c will be in processors (0,1,0,0), $(0, \sqrt{N}-1, 0, \sqrt{N}-1)$, and $(0, 0, \sqrt{N}-1, 0)$ respectively. Figure 2 shows the initial and concentrated data layout for the case when $N = 16$. The change in G_x, G_y, P_x, and P_y values between the final and initial locations of $D(a)$, $D(b)$, and $D(c)$ is shown in Table 2.

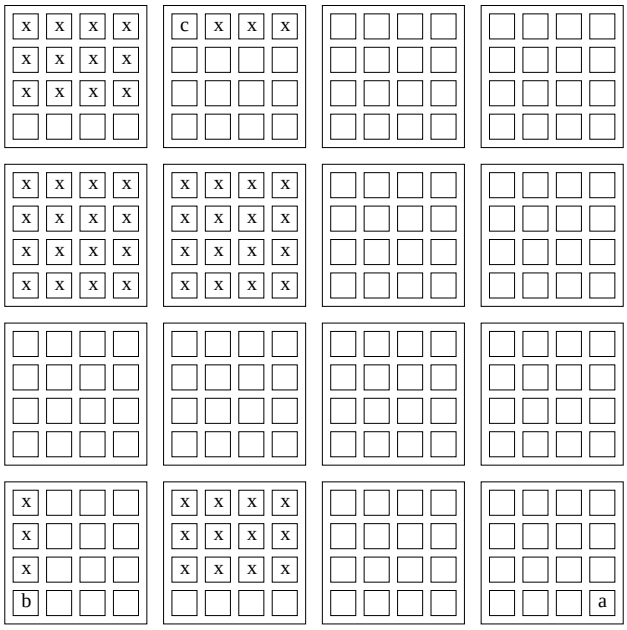

Figure 2: Data Configuration: (a) Initial; (b) Concentrated

G_x G_y	P_x P_y
0,0	$0 \le P_x < \sqrt{N}-1$, $0 \le P_y < \sqrt{N}$
0,1	$P_x = 0$, $0 \le P_y < \sqrt{N}$
$G_x = 1$, $0 \le G_y < \sqrt{N}-2$	$0 \le P_x, P_y < \sqrt{N}$
$\sqrt{N}-1$, 0	$0 \le P_x < \sqrt{N}$, $G_y = 0$
$\sqrt{N}-1$, 1	$0 \le P_x < \sqrt{N}-1$, $0 \le P_y < \sqrt{N}$
$\sqrt{N}-1$, $\sqrt{N}-1$	$\sqrt{N}-1$, $\sqrt{N}-1$

Table 1: Processors with data to concentrate

data	G_x	G_y	P_x	P_y
$D(a)$	$-(\sqrt{N}-1)+1$	$-(\sqrt{N}-1)$	$-(\sqrt{N}-1)$	$-(\sqrt{N}-1)$
$D(b)$	$-(\sqrt{N}-1)$	$+(\sqrt{N}-1)$	$-(\sqrt{N}-1)$	$+(\sqrt{N}-1)$
$D(c)$	0	0	$+(\sqrt{N}-1)$	0

Table 2: Net change in G_x, G_y, P_x, and P_y

The maximum net negative change in each of G_x, G_y, P_x, and P_y is $-(\sqrt{N}-1)$. Since a net negative change in G_x can only be overlapped with a net negative change in P_x and since $D(b)$ needs $-(\sqrt{N}-1)$ negative change in both G_x and P_x, we must make at least $2(\sqrt{N}-1)$ electronic moves that decrease the row index within a mesh. Similarly, because of $D(a)$'s requirements, at least $2(\sqrt{N}-1)$ electronic moves that increase the column index within a $\sqrt{N} \times \sqrt{N}$ mesh must be made. Turning our attention to net positive changes, we see that because of $D(b)$'s requirements there must be at least $2(\sqrt{N}-1)$ electronic moves that increase the column index. $D(c)$ requires $\sqrt{N}-1$ electronic moves that increase the row index. Since positive net moves cannot be overlapped with negative net moves, and since net moves along G_x and P_x cannot be overlapped with net moves along G_y and P_y, the concentration of the configuration of Table 1 must take at least $7(\sqrt{N}-1)$ electronic moves.

In addition to $7(\sqrt{N}-1)$ electronic moves, we need at least 2 OTIS moves to concentrate the data of Table 1. To see this consider the data initially in group (0,1). This data is in group (0,0) following the concentration. At least one OTIS move is needed to move the data out of group (0,1). A nontrivial OTIS-Mesh has ≥ 2 processors on a row of a $\sqrt{N} \times \sqrt{N}$ submesh. For such an OTIS-Mesh, at least two pieces of data must move from group (0,1) to group (0,0). A single OTIS move scatters data from group (0,1) to different groups with each data going to a different group. At least one additional OTIS move must be made to get the data back into the same group.

Therefore the concentration of the configuration of Table 1 cannot be done with fewer than 2 OTIS moves.

(b) Consider the initial configuration of Table 1. Since the shortest path between processor b and its destination processor is $4(\sqrt{N}-1)$ electronic and one OTIS move, at least that many electronic moves are made, in the worst case, by every concentration algorithm. The reason that at least 2 OTIS moves are needed to complete the concentration is the same as for *(a)*. □

2.11 Distribute

This is the inverse of the concentrate operation of Section 2.10. We start with pairs $(D_0, d_0), \ldots, (D_q, d_q)$, $d_0 < d_1 < \cdots < d_q$, in the first $q+1$ processors $0, 1, \ldots, q$ and are to route pair (D_i, d_i) to processor d_i, $0 \leq i \leq q$. The algorithm of Section 2.10 tells us how to start with pairs (D_i, i) in processor d_i, $0 \leq i \leq q$ and move them so that D_i is in i. By running this backwards, we can start with D_i in i and route it to d_i. The complexity of the distribute operation is the same as that of the concentrate operation. We have shown that the concentrate algorithm of Section 2.10 is optimal; it follows that the distribute algorithm is also optimal.

2.12 Generalize

We start with the same initial configuration as for the distribute operation. The objective is to have D_i in all processors j such that $d_i \leq j < d_{i+1}$ (set d_{q+1} to N^2-1). If we simulate the 4D mesh algorithm for generalize using the simulation strategy of [11], it takes $8(\sqrt{N}-1)$ electronic and $8(\sqrt{N}-1)$ OTIS moves to perform the generalize operation on an SIMD OTIS-Mesh. We can improve this to $8(\sqrt{N}-1)$ electronic and 2 OTIS moves if we run the generalize algorithm of [6] adapted to use OTIS moves as necessary. The outer loop of the algorithm of [6] examines processor index bits from $2p-1$ to 0 where $p = \log_2 N$. So in the first p iterations we are moving along bits of the G index and in the last p iterations along bits of the P index. On an OTIS-Mesh we would break this into two parts as below:

Step 1: Perform an OTIS move.

Step 2: Run the GENERALIZE procedure of [6] from bit $p-1$ to 0, while maintaining the original index.

Step 3: Perform an OTIS move.

Step 4: Run the GENERALIZE algorithm of [6] from bit $p-1$ to 0.

On an MIMD OTIS-Mesh the above algorithm takes $4(\sqrt{N}-1)$ electronic and 2 OTIS moves.

We can reduce the SIMD complexity to $7(\sqrt{N}-1)$ electronic and 2 OTIS moves by using a better algorithm to do the generalize operation on a 2D SIMD mesh. This algorithm uses the same observation as used by us in Section 2.10 to speed the 2D SIMD mesh concentrate algorithm; that is, of the four possible move directions, only three are possible. When doing a generalize on a 2D $\sqrt{N} \times \sqrt{N}$ mesh the possible move directions for data are to increasing row indexes and to decreasing and increasing column indexes. With this observation, the algorithm to generalize on a 2D mesh becomes:

Step 1: Move data along columns to increasing row indexes if the data is needed in a row with higher index.

Step 2: Move data along rows to increasing column indexes if the data is needed in a processor in that row with higher column index.

Step 3: Move data along rows to decreasing column indexes if the data is needed in a processor in that row with smaller column index.

The correctness of the preceding generalize algorithm can be established using the argument of Theorem 1, and its optimality follows from Theorem 2 and the fact that the distribute operation, which is the inverse of the concentrate operation, is a special case of the generalize operation.

The new and more efficient generalize algorithm may be used in Step 2 of the OTIS-Mesh generalize algorithm. It cannot be used in Step 4 because the generalize of this step requires the full capability of the code of [6] which permits data movement in all four directions of a mesh.

When we use the new generalize algorithm for Step 2 of the OTIS-Mesh generalize algorithm, we can perform a generalize on a SIMD OTIS-Mesh using $7(\sqrt{N}-1)$ electronic and 2 OTIS moves. The new algorithm is optimal for both SIMD and MIMD models. This follows from the lower bound on a concentrate operation established in Theorem 2 and the observation made above that the distribute operation, which is a special case of the generalize operation, is the inverse of the concentrate operation and so has the same lower bound.

2.13 Sorting

As was the case for the operations considered so far, an $O(\sqrt{N})$ time algorithm to sort can be obtained by simulating a similar complexity 4D mesh algorithm. For sorting a 4D Mesh, the

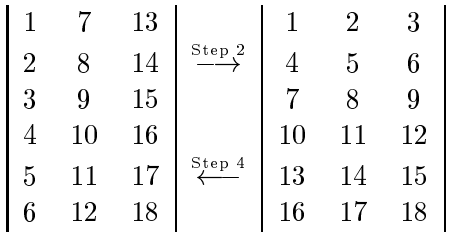

Figure 3: Row-Column Transformation of Leighton's Column Sort

algorithm of Kunde [2] is the fastest. Its simulation will sort into snake-like row-major order using $14\sqrt{N}+o(\sqrt{N})$ electronic and $12\sqrt{N}+o(\sqrt{N})$ OTIS moves on the SIMD model and $7\sqrt{N}+o(\sqrt{N})$ electronic and $6\sqrt{N}+o(\sqrt{N})$ OTIS moves on the MIMD model. To sort into row-major order, additional moves to reverse alternate dimensions are needed. This means that an OTIS-Mesh simulation of Kunde's 4D mesh algorithm to sort into row-major order will take $18\sqrt{N}+o(\sqrt{N})$ electronic and $16\sqrt{N}+o(\sqrt{N})$ OTIS moves on the SIMD model. We show that Leighton's column sort [3] can be implemented on an OTIS-Mesh to sort into row-major order using $22\sqrt{N}+o(\sqrt{N})$ electronic and $O(N^{3/8})$ OTIS moves on the SIMD model and $11\sqrt{N}+o(\sqrt{N})$ electronic and $O(N^{3/8})$ OTIS moves on the MIMD model.

Our OTIS-Mesh sorting algorithm is based on Leighton's column sort [3]. This sorting algorithm sorts an $r \times s$ array, with $r \geq 2(s-1)^2$, into column-major order using the following seven steps:

Step 1: Sort each column.

Step 2: Perform a row-column transformation.

Step 3: Sort each column.

Step 4: Perform the inverse transformation of Step 2.

Step 5: Sort each column in alternating order.

Step 6: Apply two steps of comparison-exchange to adjacent rows.

Step 7: Sort each column.

Figure 3 shows an example of the transformation of Step 2, and its inverse. Figure 4 shows a step by step example of Leighton's column sort.

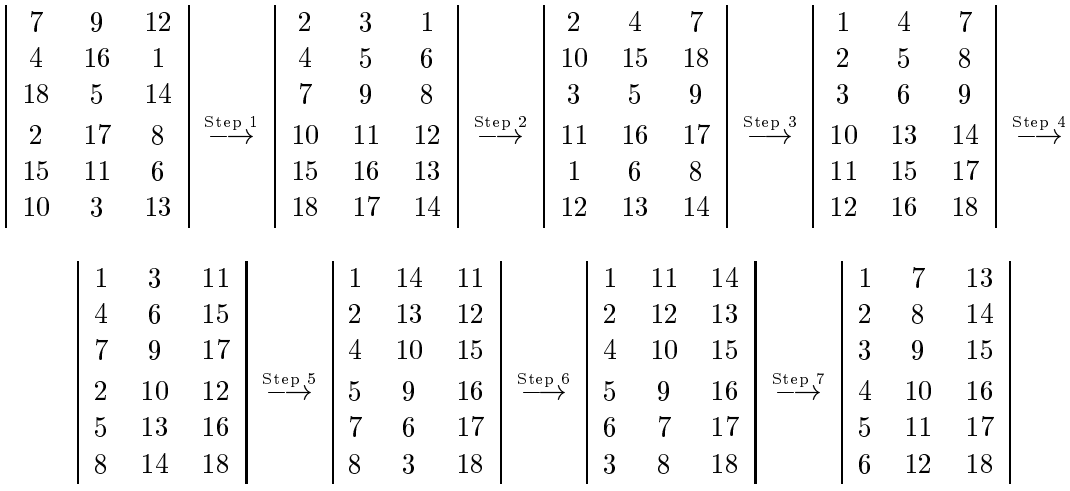

Figure 4: Example of Leighton's Column Sort

Although Leighton's column sort is explicitly stated for $r \times s$ arrays with $r \geq 2(s-1)^2$, it can be used to sort arrays with $s \geq 2(r-1)^2$ into row-major order by interchanging the roles of rows and columns. We shall do this and use Leighton's method to sort an $N^{1/2} \times N^{3/2}$ array. We interpret our N^2 OTIS-Mesh as an $N^{1/2} \times N^{3/2}$ array with G_x giving the row index and $G_y P_x P_y$ giving the column index of an element processor. We shall further subdivide G_x (G_y, P_x, P_y) into equal parts G_{x_1}, G_{x_2}, G_{x_3}, and G_{x_4} from left to right. We use $G_{x_{2-4}}$, for example, to represent $G_{x_2} G_{x_3} G_{x_4}$. Since $p = \log_2 N$, G_x has $p/2$ bits and G_{x_i} has $p/8$ bits. These notations are helpful in describing the transformations in Steps 2 and 4 of the column sort, as we use the BPC permutations of [5] to realize these transformations. A BPC permutation [5] is specified by a vector $A = [A_{p-1}, A_{p-2}, \ldots, A_0]$ where

(a) $A_i \in \{\pm 0, \pm 1, \ldots, \pm(p-1)\}$, $0 \leq i < p$ and

(b) $[|A_{p-1}|, |A_{p-2}|, \ldots, |A_0|]$ is a permutation of $[0, 1, \ldots, p-1]$.

The destination for the data in any processor may be computed in the following manner. Let $m_{p-1} m_{p-2} \ldots m_0$ be the binary representation of the processor's index. Let $d_{p-1} d_{p-2} \ldots d_0$ be that of the destination processor's index. Then,

$$d_{|A_i|} = \begin{cases} m_i & if \quad A_i \geq 0, \\ 1 - m_i & if \quad A_i < 0. \end{cases}$$

In this definition, -0 is to be regarded as < 0, while $+0$ is ≥ 0. Table 3 shows an example of the BPC permutation defined by the permutation vector $A = [-0, 1, 2, -3]$ on a 16 processor OTIS-Mesh.

Source			Destination		
Processor	(G, P)	Binary	Binary	(G, P)	Processor
0	(0,0)	0000	1001	(2,1)	9
1	(0,1)	0001	0001	(0,1)	1
2	(0,2)	0010	1101	(3,1)	13
3	(0,3)	0011	0101	(1,1)	5
4	(1,0)	0100	1011	(2,3)	11
5	(1,1)	0101	0011	(0,3)	3
6	(1,2)	0110	1111	(3,3)	15
7	(1,3)	0111	0111	(1,3)	7
8	(2,0)	1000	1000	(2,0)	8
9	(2,1)	1001	0000	(0,0)	0
10	(2,2)	1010	1100	(3,0)	12
11	(2,3)	1011	0100	(1,0)	4
12	(3,0)	1100	1010	(2,2)	10
13	(3,1)	1101	0010	(0,2)	2
14	(3,2)	1110	1110	(3,2)	14
15	(3,3)	1111	0110	(1,2)	6

Table 3: Source and destination of the BPC permutation $[\mathbf{-0, 1, 2, -3}]$ in a 16 processor OTIS-Mesh

In describing our sorting algorithm, we shall, at times, use a 4D array interpretation of an OTIS-Mesh. In this interpretation, processor (G_x, G_y, P_x, P_y) of the OTIS-Mesh corresponds to processor (G_x, G_y, P_x, P_y) of the 4D mesh. We use g_x to denote the bit positions of G_x, that is the leftmost $p/2$ bits in a processor index, g_{x_1} to represent the leftmost $p/8$ bit positions, p_y to represent the rightmost $p/2$ bit positions, $p_{y_{3-4}}$ to represent the rightmost $p/4$ bit positions, and so on. Our strategy for the sorting steps 1, 3, 5, and 7 of Leighton's method is to collect each row (recall that since we are sorting an $N^{1/2} \times N^{3/2}$ array, the column-sort steps of Leighton's method become row-sort steps) of our $N^{1/2} \times N^{3/2}$ array into an $N^{3/8} \times N^{3/8} \times N^{3/8} \times N^{3/8}$ 4D submesh of the OTIS-Mesh, and then sort this row by simulating the 4D mesh sort algorithm of [2]. This strategy translates into the following sorting algorithm:

Step 1: [Move rows of the $N^{1/2} \times N^{3/2}$ array into $N^{3/8} \times N^{3/8} \times N^{3/8} \times N^{3/8}$ 4D submeshes]

Perform the BPC permutation $P_a = [g_{x_1} g_{y_1} p_{x_1} p_{y_1} g_{x_2} g_{y_{2-4}} g_{x_3} p_{x_{2-4}} g_{x_4} p_{y_{2-4}}]$.

Step 2: [Sort each row of the $N^{1/2} \times N^{3/2}$ array]

Sort each 4D submesh of size $N^{3/8} \times N^{3/8} \times N^{3/8} \times N^{3/8}$.

Step 3: [Do the inverse of Step 1, perform a column-row transformation, and move rows into

$N^{3/8} \times N^{3/8} \times N^{3/8} \times N^{3/8}$ submeshes]

Perform the BPC permutation $P_c = [g_{x_{2-4}} g_{x_1} g_{y_2} p_{x_{2-4}} g_{y_3} p_{y_{2-4}} g_{y_4} g_{y_1} p_{x_1} p_{y_1}]$.

Step 4: [Sort each row of the $N^{1/2} \times N^{3/2}$ array]

Sort each 4D submesh of size $N^{3/8} \times N^{3/8} \times N^{3/8} \times N^{3/8}$.

Step 5: [Do the inverse of Step 1, perform a row-column transformation, and move rows into $N^{3/8} \times N^{3/8} \times N^{3/8} \times N^{3/8}$ submeshes]

Perform the BPC permutation $P_c' = [g_{x_4} g_{x_{1-3}} p_{y_2} g_{y_1} p_{x_1} p_{y_1} p_{y_3} g_{y_{2-4}} p_{y_4} p_{x_{2-4}}]$.

Step 6: [Sort each row in alternating order]

Sort each 4D submesh of size $N^{3/8} \times N^{3/8} \times N^{3/8} \times N^{3/8}$.

Step 7: [Move rows back from 4D submeshes]

Perform the BPC permutation $P_a' = [g_{x_1} g_{y_1} p_{x_1} p_{y_1} g_{x_2} g_{y_{2-4}} g_{x_3} p_{x_{2-4}} g_{x_4} p_{y_{2-4}}]$.

Step 8: Apply two steps of comparison-exchange to adjacent rows.

Step 9: [Move rows into submeshes of size $N^{3/8} \times N^{3/8} \times N^{3/8} \times N^{3/8}$]

Perform the BPC permutation $P_a = [g_{x_1} g_{y_1} p_{x_1} p_{y_1} g_{x_2} g_{y_{2-4}} g_{x_3} p_{x_{2-4}} g_{x_4} p_{y_{2-4}}]$.

Step 10: [Sort each row of the $N^{1/2} \times N^{3/2}$ array]

Sort each 4D submesh of size $N^{3/8} \times N^{3/8} \times N^{3/8} \times N^{3/8}$.

Step 11: [Move rows back from 4D submeshes]

Perform the BPC permutation $P_a' = [g_{x_1} g_{y_1} p_{x_1} p_{y_1} g_{x_2} g_{y_{2-4}} g_{x_3} p_{x_{2-4}} g_{x_4} p_{y_{2-4}}]$.

Notice that the row to 4D submesh transform is accomplished by the BPC permutation $P_a = [g_{x_1} g_{y_1} p_{x_1} p_{y_1} g_{x_2} g_{y_{2-4}} g_{x_3} p_{x_{2-4}} g_{x_4} p_{y_{2-4}}]$. Elements in the same row of our $N^{1/2} \times N^{3/2}$ array interpretation have the same G_x value; but in our 4D mesh interpretation, elements in the same $N^{3/8} \times N^{3/8} \times N^{3/8} \times N^{3/8}$ submesh have the same $G_{x_1} G_{y_1} P_{x_1} P_{y_1}$ value. P_a results in this property. To go from Step 2 to Step 3 of Leighton's method, we need to first restore the $N^{1/2} \times N^{3/2}$ array interpretation using the inverse permutation of P_a, that is, perform the BPC permutation $P_a' = [g_{x_1} g_{y_1} p_{x_1} p_{y_1} g_{x_2} g_{y_{2-4}} g_{x_3} p_{x_{2-4}} g_{x_4} p_{y_{2-4}}]$; then perform a column-row transform using BPC permutation $p_b = [g_y p_x p_y g_x]$; and finally map the rows of our $N^{1/2} \times N^{3/2}$ array into 4D submeshes of size $N^{3/8} \times N^{3/8} \times N^{3/8} \times N^{3/8}$ using the BPC permutation P_a. The three BPC permutation sequence $P_a' P_b P_a$ is equivalent to the single BPC permutation $P_c = [g_{x_{2-4}} g_{x_1} g_{y_2} p_{x_{2-4}} g_{y_3} p_{y_{2-4}} g_{y_4} g_{y_1} p_{x_1} p_{y_1}]$.

The preceding OTIS-Mesh implementation of column sort performs 6 BPC permutations, 4 4D mesh sorts, and two steps of comparison-exchange on adjacent rows. Since the sorting steps take $O(N^{3/8})$ time each (use Kunde's 4D mesh sort [2] followed by a transform from snake-like row-major to row-major), and since the remaining steps take $O(N^{1/2})$ time, we shall ignore the complexity of the sort steps.

We can reduce the number of BPC permutations from 6 to 3 as follows. First note that the P_a of Step 1 just moves elements from rows of the $N^{1/2} \times N^{3/2}$ array into $N^{3/8} \times N^{3/8} \times N^{3/8} \times N^{3/8}$ 4D submeshes. For the sort of Step 2, it doesn't really matter which $N^{3/2}$ elements go to each 4D submesh as the initial configuration is an arbitrary unsorted configuration. So we may eliminate Step 1 altogether. Next note that the BPC permutations of Steps 7 and 9 cancel each other and we can perform the comparison-exchange of Step 8 by moving data from one $N^{3/8} \times N^{3/8} \times N^{3/8} \times N^{3/8}$ 4D submesh to an adjacent one and back in $O(N^{3/8})$ time.

With these observations, the algorithm to sort on an OTIS-Mesh becomes:

Step 1: Sort in each subarray of size $N^{3/8} \times N^{3/8} \times N^{3/8} \times N^{3/8}$

Step 2: Perform the BPC permutation P_c.

Step 3: Sort in each subarray.

Step 4: Perform the BPC permutation P_c'.

Step 5: Sort in each subarray.

Step 6: Apply two steps of comparison-exchange to adjacent subarrays.

Step 7: Sort in each subarray.

Step 8: Perform the BPC permutation P_a'.

Using the BPC routing algorithm of [10], the three BPC permutations can be done using $36\sqrt{N}$ electronic and $3\log_2 N + 6$ OTIS moves on the SIMD model and $18\sqrt{N}$ electronic and $3\log_2 N + 6$ OTIS moves on the MIMD model. A more careful analysis based on the development in [5] and [10] reveals that the permutations P_a', P_c, and P_c' can be done with $28\sqrt{N}$ electronic and $\log_2 N + 6$ OTIS moves on the SIMD model and $14\sqrt{N}$ electronic and $3\log_2 N + 6$ OTIS moves on the MIMD model. By using $p_a' = [g_{x_1} g_{y_1} p_{x_1} p_{y_1} g_{x_2} g_{y_2} p_{x_2} p_{y_2} g_{x_3} g_{y_3} p_{x_3} p_{y_3} g_{x_4} g_{y_4} p_{x_4} p_{y_4}]$, $p_c = [g_{x_{2-4}} g_{x_1} g_{y_{2-4}} g_{y_1} p_{x_{2-4}} p_{x_1} p_{y_{2-4}} p_{y_1}]$, and $p_c' = [g_{x_4} g_{x_{1-3}} g_{y_4} g_{y_{1-3}} p_{x_4} p_{x_{1-3}} p_{y_4} p_{y_{1-3}}]$, the permutation

cost becomes $22\sqrt{N}$ electronic and $\log_2 N+5$ OTIS moves on the SIMD model and $11\sqrt{N}$ electronic and $\log_2 N+5$ OTIS moves on the MIMD model. The total number of moves is thus $22\sqrt{N}+O(N^{3/8})$ electronic and $O(N^{3/8})$ OTIS moves on the SIMD model and $11\sqrt{N}+O(N^{3/8})$ electronic and $O(N^{3/8})$ OTIS moves on the MIMD model. This is superior to the cost of the sorting algorithm that results from simulating the 4D row-major mesh sort of Kunde [2].

2.14 Random Access Read (RAR)

In a random access read (RAR) [8] processor I wishes to read data variable D of processor d_I, $0 \leq I < N^2$. The steps suggested in [8] for this operation are:

Step 0: Processor I creates a triple (I, D, d_I) where D is initially empty.

Step 1: Sort the triples by d_I.

Step 2: Processor I checks processor $I+1$ and deactivates if both have triples with the same third coordinate.

Step 3: Rank the remaining processors.

Step 4: Concentrate the triples using the ranks of Step 3.

Step 5: Distribute the triples according to their third coordinates.

Step 6: Load each triple with the D value of the processor it is in.

Step 7: Concentrate the triples using the ranks in Step 3.

Step 8: Generalize the triples to get the configuration we had following Step 1.

Step 9: Sort the triples by their first coordinates.

Using the SIMD model the RAR algorithm of [8] take $79(\sqrt{N}-1)$ electronic moves and $O(N^{3/8})$ OTIS moves. On the MIMD model, it takes $45(\sqrt{N}-1)$ electronic $O(N^{3/8})$ OTIS moves.

2.15 Random Access Write (RAW)

Now processor I wants to write its D data to processor d_I, $0 \leq I < N^2$. The steps in the RAW algorithm of [8] are:

Step 0: Processor I creates the tuple $(D(I), d_I)$, $0 \leq I < N^2$.

Step 1: Sort the tuples by their second coordinates.

Step 2: Processor I deactivates if the second coordinate of its tuple is the same as the second coordinate of the tuple in $I+1$, $0 \leq I < N^2 - 1$.

Step 3: Rank the remaining processors.

Step 4: Concentrate the tuples using the ranks of Step 3.

Step 5: Distribute the tuples according to their second coordinates.

Step 2 implements the arbitrary write method for a concurrent write. In this, any one of the processors wishing to write to the same location is permitted to succeed. The priority model may be implemented by sorting in Step 1 by d_I and within d_I by priority. The common and combined models can also be implemented, but with increased complexity.

On the SIMD model, an RAW takes $43(\sqrt{N}-1)$ electronic and $O(N^{3/8})$ OTIS moves while on the MIMD model, it takes $26(\sqrt{N}-1)$ electronic and $O(N^{3/8})$ OTIS moves.

3 Conclusion

We have developed OTIS-Mesh algorithms for the basic parallel computing algorithms of [8]. Our algorithms run faster than the simulation of the fastest algorithms known for 4D meshes. Table 4 summarizes the complexities of our algorithms and those of the corresponding ones obtained by simulating the best 4D-mesh algorithms. Note that the worst case complexities are listed for the broadcast and window broadcast operation, and that of the case when $\sqrt{N}$ is even is presented for the data sum operation on the MIMD model. Also, the complexities listed for circular shift, data accumulation, and adjacent sum assume that the shift distance is $\leq \sqrt{N}/2$ on the MIMD model. Table 4 gives only the dominating $\sqrt{N}$ terms for sorting. Our algorithms for data broadcast, data sum, concentrate, distribute, and generalize are optimal.

References

[1] A. Krishnamoorthy, P. Marchand, F. Kiamilev, and S. Esener. Grain-size considerations for optoelectronic multistage interconnection networks. *Applied Optics*, 31(26), Sept. 1992.

[2] M. Kunde. Routing and sorting on mesh-connected arrays. In *Proceedings of the 3rd Agean Workshop on Computing: VLSI Algorithms and Architectures, Lecture Notes on Computer Science*, volume 319, pages 423–433. Springer Verlag, 1988.

[3] T. Leighton. Tight bounds on the complexity of parallel sorting. *IEEE Transactions on Computers*, C-34(4):344–354, Apr. 1985.

[4] G. C. Marsden, P. J. Marchand, P. Harvey, and S. C. Esener. Optical transpose interconnection system architectures. *Optics Letters*, 18(13):1083–1085, July 1 1993.

[5] D. Nassimi and S. Sahni. An optimal routing algorithm for mesh-connected parallel computers. *Journal of the Association for Computing Machinery*, 27(1):6–29, Jan. 1980.

[6] D. Nassimi and S. Sahni. Data broadcasting in SIMD computers. *IEEE Transactions on Computers*, C-30(2):101–107, Feb. 1981.

[7] S. Rajasekaran and S. Sahni. Randomized routing, selection, and sorting algorithms on the OTIS-Mesh optoelectronic computer. manuscript, 1997.

[8] S. Ranka and S. Sahni. *Hypercube Algorithms with Applications to Image Processing and Pattern Recognition*. Springer-Verlag, 1990.

[9] S. Sahni and C.-F. Wang. BPC permutations on the OTIS-Hypercube optoelectronic computer. Technical report, CISE Department, University of Florida, 1997.

[10] S. Sahni and C.-F. Wang. BPC permutations on the OTIS-Mesh optoelectronic computer. In *Proceedings of the fourth International Conference on Massively Parallel Processing Using Optical Interconnections (MPPOI'97)*, pages 130–135, 1997.

[11] F. Zane, P. Marchand, R. Paturi, and S. Esener. Scalable network architectures using the optical transpose interconnection system (OTIS). In *Proceedings of the second International Conference on Massively Parallel Processing Using Optical Interconnections (MPPOI'96)*, pages 114–121, 1996.