Sorting n^2 Numbers On $n \times n$ Meshes*

Madhusudan Nigam and Sartaj Sahni

University of Florida

Gainesville, FL 32611

Technical Report 92-34

ABSTRACT

We show that by folding data from an $n \times n$ mesh onto an $n \times (n/k)$ submesh, sorting on the submesh, and finally unfolding back onto the entire $n \times n$ mesh it is possible to sort on bidirectional and strict unidirectional meshes using a number of routing steps that is very close to the distance lower bound for these architectures. The technique may also be applied to reconfigurable bus architectures to obtain faster sorting algorithms.

Keyword and Phrases

Sorting, mesh architectures, distance lower bound

This research was supported, in part, by the National Science Foundation under grant MIP-9103379.

1 Introduction

In this paper, we are concerned with sorting n^2 data elements using an $n \times n$ mesh connected parallel computer. The initial and final configurations have one data element in each of the $n \times n$ processors (say in the A variable of each processor). In the final configuration the data elements are sorted in snake-like row-major order. This problem has been extensively studied for mesh architectures (see for e.g. [THOM77], [NASS79], [KUMA83], [LEIG85], [SCHN86], [SCHER89], [MARB88]). While all of these studies consider SIMD meshes, they differ in the permissible communication patterns. Thompson and Kung [THOM77] consider a *strict unidirectional* model in which all processors that simultaneously transfer data to a neighbor processor do so to the same neighbor. That is, all active processors transfer data to their north neighbor, or all to their south neighbor, etc. Using this model, Thompson and kung [THOM77] have developed a sorting algorithm with comlexity $6nt_r + nt_c$ + low order terms, where t_r is the time needed to transfer one data element from a processor to its neighbor and t_c is the time needed to compare two data elements that are in the same processor. In the sequel, we omit the low order terms.

In the *bidirectional model*, we assume there are two links between every pair (P,Q) of neighbors. As a result, P can send a data element to Q at the same time that Q sends one to P. Using this model, Schnorr and Shamir [SCHN86] have developed a sorting algorithm with complexity $3nt_r + 2nt_c$. In ths same paper, Schnorr and Shamir "prove" that $3n$ routing steps are needed by every sorting algorithm for an even stronger mesh model, each processor can read the entire contents of the memories of its (up to) four neighbors in time t_r and the internal processor computations are free. For this stronger mesh model, they show that every sorting algorithm must take $3nt_r$ time by first showing that input data changes in the lower right part of the mesh cannot affect the values in the top left corner processor until time equal to the distance, d, between the top left corner processor and the nearest of these lower right processors. Next, they argue that by changing the lower right input data, they can change the final position of the data in the top left corner processor by distance $n-1$. As a result the sort must take at least $(d+n-2)t_r$ time.

The fallacy is that in the first $d-1$ steps, we may have made several copies of the input data and it may no longer be necessary to route the data from the top left corner processor to the final destination processor. In fact, we show that one can sort in $2nt_r$ time using the stronger mesh model.

Park and Balasubramanium [PARK87,90] have considered a related sorting problem for the strict unidirectional model. In this the n^2 elements to be sorted are initially stored two to a processor on an $n \times n/2$ mesh. The final sorted configuration also has two elements per processor. The time complexity of their algorithm is $4nt_r + nt_c$. This represents an improvement of $2nt_r$ over Thompson and Kung's algorithm provided the two element per processor inpu/output configuration is what we desire. If we desire the one element per processor configuration (for example, the data to be sorted is the result of a computation that produces this configuration and the sorted data is to be used for further computation), then it is necessary to first fold the data to get the two element per processor configuration, then sort using the algorithm of [PARK87, 90], and finally unfold to get the desired one element per processor final configuration. The folding can be done in $n/2\ t_r$ time as below (see also Figure 1(a)).

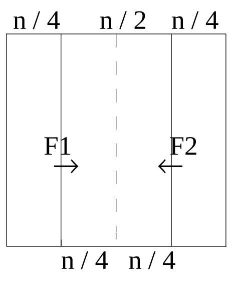
F1: The left $n/4$ columns shift their data $n/4$ columns to right.

F2: The right $n/4$ columns shift their data $n/4$ columns to the left.

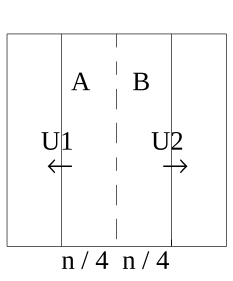
The unfolding can also be done in $n/2\ t_r$ time using the two steps:

U1: Unfold the $n/4$ columns labeled A in Figure 1(b)

U2: Unfold the $n/4$ columns labeled B in Figure 1(b)



(a) Folding



(b) Unfolding

Figure 1 Folding and Unfolding

To unfold A, we use the pipelined process described by the example of Figure 2. B is unfolded in a similar way. The total time for the sort is therefore $5nt_r + nt_c$ which is nt_r less than that of [THOM77]. The improvement is slightly more if we consider the (non-strict) unidirectional model in which there is a single link between each pair of neighbor processors and data can be transferred, in parallel, along all links (however, if P and Q are neighbors, when P sends data to Q, Q cannot send to P). In this model, steps $F1$ and $F2$ of folding can be done in parallel. Similarly, we can do $U1$ and $U2$ in parallel. The total sort time now becomes $4.5nt_r + nt_c$.

In section 2, we show that the Schnorr/Shamir algorithm of [SCHN86] can be modified to sort on unidirectional meshes using the same number of routes as above. The number of comparison steps is, however, larger. The modified Schnorr/Shamir algorithm of section 2 runs in $2.5nt_r + 3nt_c$ time on an $n \times n$ bidirectional mesh. The number of routes is therefore less than the $3n$ *lower bound* established in [SCHN86]. The algorithm of section 2 folds data onto an $n \times n/2$ submesh. By folding onto smaller submeshes, i.e., onto $n \times n/k$ submeshes for $k > 2$, the number of routes can be reduced further. In fact for bidirectional and strict unidirectional meshes we can come very close to the distance lower bound of $2n-2$ and $4n-2$, respectively. This is done in section 3. In section 4, we discuss the practical implications of the folding-unfolding approach to sorting. In section 5, we show how the sort algorithms of sections 2 and 3 may be simulated on $n \times n$ reconfigurable bus architectures.

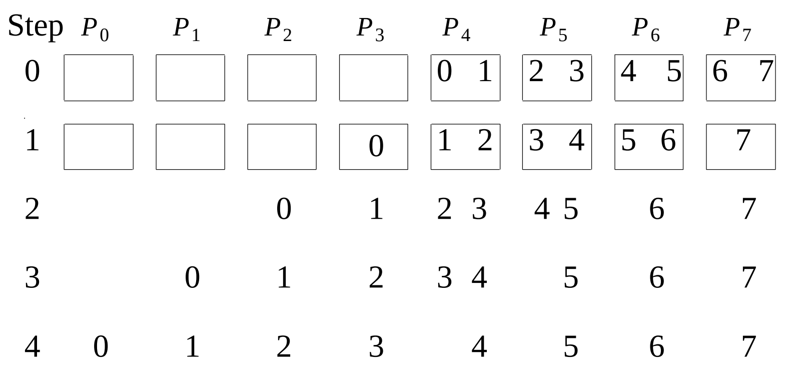


Figure 2 Unfolding one row of A

2 Modified Schnorr/Shamir Algorithm

The sorting algorithm of Schnorr and Shamir [SCHN86] is given in Figure 3. This algorithm uses the following terms and assumes that $n = 2^{4q}$ for some integer q.

1. *Block*... an $n^{3/4} \times n^{3/4}$ submesh formed by a natural tiling of an $n \times n$ mesh with $n^{3/4} \times n^{3/4}$

tiles (Figure 4(a)).

2. *Vertical Slice*... an $n \times n^{3/4}$ submesh formed by a natural tiling of an $n \times n$ mesh with $n \times n^{3/4}$ tiles (Figure 4(b)).

3. *Snake*... the $1 \times n^2$ vector along the snake-like order in the $n \times n$ mesh (Figure 4(c)).

4. *Even-odd transposition sort* [KNUT73]...n elements are sorted in n steps. In the odd steps each element in an odd position is compared with the element in the next even position and exchanged if larger. In the even steps, each element in an even position is compared with the element in the next odd position and exchanged if greater.

5. $n^{1/4}$*-way unshuffle* ... Let $k = 1/4 \log_2 n = q/4$. Data in column j of the mesh is moved to column j'. Let $j_{q-1} \ldots j_0$ be the binary representation of j. Then $j_{k-1} \ldots j_0 \, j_{q-1} \ldots j_k$ is the binary representation of j'. The unshuffle distributes the $n^{3/4}$ columns of each vertical slice to the $n^{1/4}$ vertical slices in a round robin manner.

Step 1: Sort all the blocks into snake-like row-major order.
Step 2: Perform a $n^{1/4}$-way unshuffle along all the rows of the array.
Step 3: Sort all the blocks into snake-like row-major order.
Step 4: Sort all the columns of the array downwards.
Step 5: Sort all the vertical slices into snake-like row-major order.
Step 6: Sort all the rows of the array into alternating left-to-right and right-to-left order.
Step 7: Perform $2n^{3/4}$ steps of even-odd transposition sort along the snake.

Figure 3 Sorting algorithm of Schnorr and Shamir

The correctness of the algorithm is established in [SCHN86]. As pointed out in [SCHN86], steps 1,3,5,7 take $O(n^{3/4})$ time and are dominated by steps 2,4,6. Steps 4 and 6 are done using n steps of even-odd transposition sort each while step 2 is done by simulating even-odd transposition sort on the fixed input permutation π^{-1} where π is the desired unshuffle permutation. Steps 4 and 6 take $nt_r + nt_c$ time each on a bidirectional

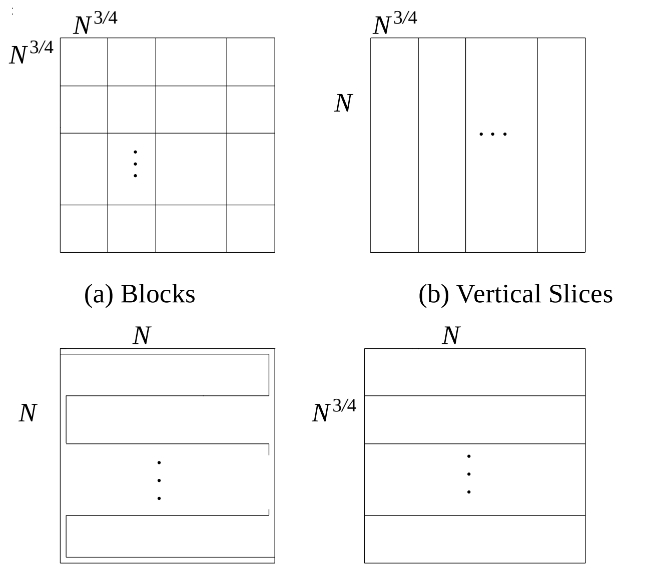
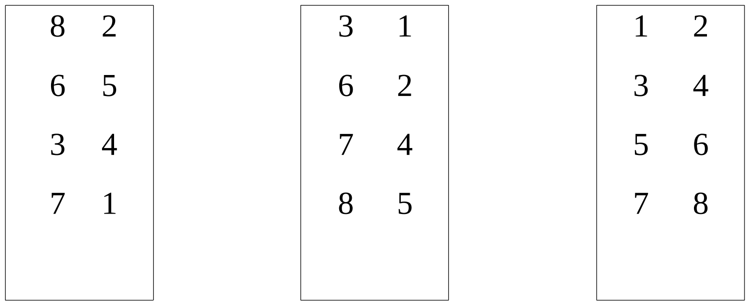


Figure 4 Definitions of Blocks and Slices

mesh and $2nt_r + nt_c$ on a unidirectional (whether strict or not) one. Step 2 takes nt_r time on a bidirectional mesh and $2nt_r$ on a unidirectional one. The total time for the algorithm of Schnorr and Shamir is therefore $3nt_r + 2nt_c$ on a bidirectional mesh and $6nt_r + 2nt_c$ on a unidirectional (strict or nonstrict) mesh. On the stronger mesh model the time is $3nt_r$ as t_c is zero.

We may modify the algorithm of Schnorr and Shamir so that it works on an $n \times n/2$ mesh with two elements per processor. In this modification, we essentially simulate the algorithm of Figure 3 using half as many processors. The run time for steps 1,3,5 and 7 increases but is still $O(n^{3/4})$. Step 6 takes $n/2\, t_r + nt_c$ on a bidirectional mesh as data routing is needed for only half of the steps of even-odd transposition sort and $nt_r + nt_c$ on a unidirectional mesh. Step 4 takes $2nt_r + 2nt_c$ on a bidirectional mesh and $4nt_r + 2nt_c$ on a unidirectional mesh as in each of the n steps of even-odd transposition sort, both the elements in a processor need to be routed to the neighbor processor. This can be reduced to $nt_r + 2nt_c$ and $2nt_r + 2nt_c$, respectively, by regarding the $2n$ elements in a processor column as a single column and sorting this into row major order using $2n$ steps of even-odd transposition sort (Figure 5). Now, a single element from each processor is to be routed on each of n steps and no data routing is needed on the remaining n steps.



(a) Processor column two elements per processor (b) Sort each column downward (c) Sort as one column

Figure 5 Modified Step 4

To establish the correctness of the sorting algorithm with step 6 changed to sort pairs of columns as though they were a single column, we use the zero-one principle [KNUT73]. This was also used by Schnorr and Shamir to establish the correctness of their unmodified algorithm. Here, we assume that the input data consists solely of zeroes and ones. Since we have not changed steps 1-3 of their algorithm, the proofs of parts 1,2, and 3 of their Theorem 3 are still valid. We shall show that part 4 remains true following the execution of the modified step 4. Since the proof of parts 5,6, and 7 rely only part 4 of the theorem, these are valid for the modified algorithm. Hence, the modified algorithms is correct.

Define a *horizontal slice* to be an $n^{3/4} \times n$ submesh obtained by a natural tiling of an $n \times n$ mesh using tiles of size $n^{3/4} \times n$ (Figure 4(d)). Following step 3 of the sorting algorithm, the maximum difference d_3 between the number of zeroes in any two columns of a horizontal slice is at most 2 [SCHN86]. We need to show (i) that following the modified step 4, the maximum difference d_4' between the number of zeroes in any two columns (a column is an $n \times 1$ vector with n elements) of the mesh is at most $2n^{1/4}$, and (ii) the maximum difference between the number of zeroes in any two vertical slices is atmost $n^{1/2}$. The proof of (ii) is same as that in [SCHN86] as our modification of step 4 does not move data between vertical slices. For (i), consider any two columns of n elements each. If these two columns reside in the same column of processors, then the maximum difference between the number of zeroes between two columns is 1 as the two columns have been sorted into row major order regarding them as a single $2n$ element column. Suppose the two element columns reside in two different processor columns. These two processor columns contain 4 element columns A,B,C,D. Let a,b,c,d respectively be the number of zeroes in these four columns prior to the execution of the modified step 4. From the proof of Theorem 3, part 4 [SCHN86], we know that $|x-y| \le 2n^{1/4}$ where $x,y \in \{a,b,c,d\}$.

Let a',b',c',d' be the number of zeroes in column A,B,C,D following the modified step 4. It is easy to see that $a' = \lceil (a+b)/2 \rceil$, $b' = \lfloor (a+b)/2 \rfloor$, $c' = \lceil (c+d)/2 \rceil$, $d' = \lfloor (c+d)/2 \rfloor$. We need to show that $|x-y| \le 2\, n^{1/4}$ where $x,y \in \{a',b',c',d'\}$.

Without loss of generality (wlog), we may assume that $|b'-c'| =$ max $\{|x-y| \mid x,y \in \{a',b',c',d'\}\}$ (note that when $x,y \in \{a', b'\}$ or $x,y \in \{c',d'\}$, $|x-y| \le 1$) and that $b' > c'$. $b'-c' \le \lceil (a+b)/2 \rceil - \lfloor (c+d)/2 \rfloor$. Again, wlog we may assume that $c \le d$. So, $b'-c' \le \lceil (a+b)/2 \rceil - c$. Since, $|b-c| \le 2n^{1/4}$ and $|a-c| \le 2n^{1/4}$, $a+b \le 2c + 4n^{1/4}$. So, $b'-c' \le c + 2n^{1/4} - c = 2n^{1/4}$.

Our modified Schnorr/Shamir algorithm for $n \times n$ meshes is stated in Figure 6. It combines the folding and unfolding steps discussed in the introduction and the Schnorr/Shamir modified algorithm for $n \times n/2$ meshes described above.

Theorem 1: The sorting algorithm of Figure 6 is correct.

Proof: Follows directly from our earlier discussion that established the correctness of steps 1 through 7 as a sorting algorithm for an $n \times n/2$ mesh with two elements per processor. □

For the complexity analysis, we note that steps 1,3,5, and 7 take O($n^{3/4}$) time and are dominated by the remaining steps which take O(n) time each. Since we are ignoring low order terms, we need be concerned only with steps 0, 2, 4, 6, and 8. As noted earlier, steps 2, 4 and 6, respectively, take $n/2\, t_r$, $nt_r + 2nt_c$, and $n/2\, t_r + nt_c$ on a bidirectional mesh. Steps 0 and 8 each take $n/4\, t_r$ on a bidirectional mesh. So, the complexity of the sort algorithm on a bidirectional mesh is $2.5n\, t_r + 3n\, t_c$. Since t_c is zero on the stronger mesh model, the sort time for this model is $2.5n\, t_r$ (note that the algorithm of [SCHN86] has a run time of $3n\, t_r$).

On a (nonstrict) unidirectional mesh, the times for steps 0,2,4,6, and 8 are, respectively, $n/4\, t_r$, $n\, t_r$, $2n\, t_r + 2n\, t_c$, $n\, t_r + n\, t_c$, $n/4\, t_r$. The total time for this model is therefore $4.5n\, t_r + 3n\, t_c$. On a strict unidirectional mesh, the times for steps 0,2,4,6, and 8 are respectively, $n/2\, t_r$, $n\, t_r$, $2n\, t_r + 2n\, t_c$, $n\, t_r + n\, t_c$, $n/2\, t_r$ and the total time is $5n\, t_r + 3n\, t_c$.

Step 0: Fold the leftmost and rightmost $n/4$ columns so that the n^2 elements to be sorted are in the middle $n \times n/2$ submesh with each processor in this submesh having two elements.

Step 1: Sort all $n^{3/4} \times n^{3/4}$ blocks of data into snake-like row-major order. Note that each block of data is in an $n^{3/4} \times (1/2\ n^{3/4})$ submesh.

Step 2: Perform an $n^{1/4}$-way unshuffle along each row of n elements. Note that each row of n elements is in a row of the middle $n \times (1/2\ n)$ submesh.

Step 3: Repeat step 1.

Step 4: Sort the $2n$ elements in each column of the middle $n \times n/2$ submesh into row-major order.

Step 5: Sort the elements in each $n \times (1/2\ n^{3/4})$ submesh into snake-like row-major order.

Step 6: Sort all rows of the middle $n \times n/2$ submesh into alternating left-to-right and right-to-left order.

Step 7: Perform $2n^{3/4}$ steps of even-odd transposition sort along the snake of the middle $n \times n/2$ submesh.

Step 8: Unfold the middle $n \times n/2$ submesh.

Figure 6 Sorting algorithm for $n \times n$ mesh.

3 Further Enhancements

In the stronger mesh model of Schnorr and Shamir a processor can read the entire memory of all its neighbors in unit time. This implies that the routing time is independent of message length. Let T_r denote the time needed to send a message of arbitrary length to a neighbor processor. We may generalize the sorting algorithm of Figure 6 to the case when each processor has k elements. Now, to sort a row or column of data, we use neighborhood sort [BAUD78] which is a generalization of even-odd transposition sort. Suppose that m processors have k elements each. The mk elements are sorted in m steps. In the even steps, the elements in each even processor are merged with those in the next odd processor. The even processor gets the smaller k and the odd one the larger k. In odd steps, the k elements in each odd processor are merged with the k elements in the next even processor. The smaller k elements remain with the odd processor and the larger k

with the even one. Note that when $k = 1$ neighborhood sort is identical to even-odd transposition sort.

Our generalization of Figure 6 is given in Figure 7. We require that k be a power of 2. As in the case of the Schnorr-Shamir algorithm, we assume $n = 2^{4q}$.

Step 0: Fold the leftmost and the rightmost $n(k-1)/(2k)$ columns into middle n/k columns so that the n^2 elements to be sorted are in the middle $n \times n/k$ submesh with each processor in this submesh having k elements. Sort the k elements in each processor of the middle $n \times n/k$ submesh.

Step 1: Sort all $n^{3/4} \times n^{3/4}$ blocks of data into snake-like row-major order. Note that each block of data is in an $n^{3/4} \times n^{3/4}/k$ submesh.

Step 2: Perform an $n^{1/4}$-way unshuffle along each row of n elements. Note that each row of n elements is in a row of the middle $n \times n/k$ submesh.

Step 3: Repeat step 1.

Step 4: Sort the kn elements in each column of the middle $n \times n/k$ submesh into row-major order. Use neighborhood sort.

Step 5: Sort the elements in each $n \times (n^{3/4}/k)$ submesh into snake-like row-major order.

Step 6: Sort all rows of the middle $n \times n/k$ submesh into alternating left-to-right and right-to-left order.

Step 7: Perform $2n^{3/4}$ steps of even-odd transposition sort along the snake of the middle $n \times n/k$ submesh.

Step 8: Unfold the middle $n \times n/k$ submesh.

Figure 7 Generalized sorting algorithm

Theorem 2: The generalized sorting algorithm is correct.

Proof: Similar to that of Theorem 1. □

For the complexity analysis, we may again ignore the odd steps as these run in O($n^{3/4}$) time for any fixed k. The folding and and unfolding of steps 0 and 8 each take $n(k-1)/(2k)\ T_r$ on bidirectional and nonstrict unidirectional meshes and $n(k-1)/k\ T_r$ on strict unidirectional meshes. The sorting of step 0 takes O($k\log k$) time. Step 2 takes $n/k\ T_r$ on bidirectional meshes and $2n/k\ T_r$ on unidirectional ones. Step 4 requires n steps of neighborhood sort. Each merge of two sets of k elements takes atmost $2k\ t_c$ time (actually atmost $2k-1$ comparisons are needed). So, the step 4 time is $n\ T_r + 2kn\ t_c$ for bidirectional meshes and $2n\ T_r + 2kn\ t_c$ for unidirectional meshes. The time for step 6 is $n/k\ T_r + 2n\ t_c$ for bidirectional meshes and $2n/k\ T_r + 2n\ t_c$ for unidirectional ones.

The total sorting time (ignoring the time to sort k elements in step 0) for a bidirectional mesh is therefore $(2n + n/k)\ T_r + 2n(k+1)\ t_c$. For the model of Schnorr and Shamir [SCHN87], $t_c = 0$ and the time becomes $(2n + n/k)\ T_r$. For large values of k, this approximates to $2n\ T_r$. Since $(2n-2)\ T_r$ is the distance lower bound for sorting on Schnorr and Shamir's model, the generalized sorting algorithm of Figure 7 is near optimal for large k. The sorting times on non-strict and strict unidirectional meshes are $(3n + 3n/k)\ T_r + 2n(k+1)\ t_c$ and $(4n + 2n/k)\ T_r + 2n(k+1)\ t_c$. Since $(4n-2)$ is the distance lower bound for the strict unidirectional model, our algorithm is near optimal for large k for this model too.

The s^2-way merge sorting algorithm of Thompson and Kung [THOM77] may be similarly generalized to sort n^2 elements stored k to a PE in an $n \times (n/k)$ mesh configuration. The resulting sort has a complexity that is almost identical to that of Figure 7. However, Figure 7 is conceptually much simpler.

4 Practical Implications

Suppose we are to sort n^2 elements, that are, initially distributed one to a PE on an $n \times n$ bidirectional mesh. The final configuration is also to have one element per PE. On realistic computers, the time to transfer small packets of data between adjacent processors is dominated by the setup time. For instance the time to transfer N bytes of data between adjacent processors of an NCube1 hypercube is $(446.7 + 2.4\,N)$ms. Therefore, it is reasonable to assume that the data transfer time is independent of packet size for small k and small element size. Furthermore, $t_R \gg t_c$ on most commercial computers. For instance, $t_R \approx 40\ t_c$ on the NCube1. Table 1 gives the value of our complexity functions for the different bidirectional sort algorithms and different k. When $t_R \approx 40\ t_c$ we can expect to get best performance using the algorithm of section 3 with $k = 4$. For $t_R \approx 10\ t_c$, the algorithm of section 2 is the best of the three. The original Schnorr/Shamir algorithm will perform best only when $t_R \leq 2\ t_c$.

	Schnorr & Shamir $3n\ t_R + 2n\ t_c$	**Section 2** $2.5n\ t_R + 3n\ t_c$	**Section 3** $(2n + \frac{n}{k})\ t_R + 2n\,(k+1)\ t_c$			
k	**1**	**2**	**3**	**4**	**5**	**6**
$t_R = 40\ t_c$	$122n\ t_c$	$103n\ t_c$	$101.33n\ t_c$	$100n\ t_c$	$100n\ t_c$	$100.67n\ t_c$
$t_R = 10\ t_c$	$32n\ t_c$	$28n\ t_c$	$31.3n\ t_c$	$32.5n\ t_c$	$34n\ t_c$	$35.67n\ t_c$
$t_R = 5\ t_c$	$17n\ t_c$	$15.5n\ t_c$	$19.66n\ t_c$	$21.25n\ t_c$	$23n\ t_c$	$24.83n\ t_c$
$t_R = 2\ t_c$	$8n\ t_c$	$8n\ t_c$	$12.67n\ t_c$	$14.5n\ t_c$	$16.4n\ t_c$	$18.33n\ t_c$

Table 1 Comparison of bidirectional sort algorithms

5 Reconfigurable Meshes with Buses

The sorting algorithms described here may be simulated by parallel computers in the reconfigurable mesh with buses (RMB) family. We consider only two members of the RMB family: RMESH and PARBUS.

In an RMESH [MILL88abc], we have a bus grid with an $n \times n$ arrangement of processors at the grid points (see Figure 8 for a 4x4 RMESH). Each grid segment has a switch on it which enables one to break the bus, if desired, at that point. When all switches are closed, all n^2 processors are connected by the grid bus. The switches around a processor can be set by using local information. If all processors disconnect the switch on their north, then we obtain row buses (Figure 9). Column buses are obtained by having each processor disconnect the switch on its east (Figure 10). In the exclusive write model two processors that are on the same bus cannot simultaneously write to that bus. In the concurrent write model several processors may simultaneously write to the same bus. Rules are provided to determine which of the several writers actually succeeds (e.g., arbitrary, maximum, exclusive or, etc.). Notice that in the RMESH model it is not possible to simultaneously have n disjoint row buses and n disjoint column buses that, respectively, span the width and height of the RMESH. It is assumed that processors on the same bus can communicate in O(1) time. RMESH algorithms for fundamental data movement operations and image processing problems can be found in [MILL88abc, MILL91ab, JENQ91abc].

An $n \times n$ PARBUS (Figure 11) [WANG90] is an $n \times n$ mesh in which the interprocessor links are bus segments and each processor has the ability to connect together arbitrary subsets of the four bus segments that connect to it. Bus segments that get so connected behave like a single bus. The bus segment interconnections at a proccessor are done by an internal four port switch. If the upto four bus segments at a processor are labeled N (North), E (East), W (West), and S (South), then this switch is able to realize any set, A = $\{A_1, A_2\}$, of connections where $A_i \subseteq$ {N,E,W,S}, $1 \le i \le 2$ and the A_i's are disjoint. For example A = {{N,S}, {E,W}} results in connecting the North and South segments together and the East and West segments together. If this is done in each processor, then we get, simultaneously, disjoint row and column buses (Figure 12 and 13). If A = {{N,S,E,W},ϕ}, then all four bus segments are connected. PARBUS algorithms for a variety of applications can be found in [MILL91a, WANG90ab, LIN92, JANG92]. Observe that in an RMESH the realizable connections are of the form A = $\{A_1\}$, $A_1 \subseteq$ {N,E,W,S}.

The RMESH requires two steps to simulate a unit route along a row or column of a unidirectional mesh. The PARBUS can simulate such a route in one step using the bus configurations of Figures 12 and 13. Additionally, the folding and unfolding into/from n/k columns can be done in $(k-1)t_r$ time by having each group of k adjacent columns fold into the leftmost column in the group. Now, the n/k columns that contain the data are not adjacent. However, this does not result in any inefficiency when simulating steps 1-7 as row buses to connect the columns are easily established. With this modification to the algorithms of Figures 6 and 7, the sort time for an $n \times n$ RMESH becomes $(8n + 2)\, t_r + 3n\, t_c$, $(4n + 8n/k + 2k - 2)\, T_r + 2n(k+1)\, t_c$ and for an $n \times n$ PARBUS becomes $(4n + 2)\, t_r + 3n\, t_c$, $(2n + 4n/k + 2k - 2)\, T_r + 2n(k+1)\, t_c$.

6 Conclusion

We have shown that the lower bound for the number of routes needed to sort on an $n \times n$ bidirectional mesh that was established in [SCHN86] is incorrect. Furthermore, we have provided algorithms that sort using fewer routes than the lower bound of [SCHN86]. In fact, the algorithm of section 3 is able to sort using a number of routes that is very close to the distance lower bound for bidirectional as well as strict unidirectional meshes.

The fold/sort/unfold algorithms developed here have practical implications to sorting on a mesh. The advantages of this technique when applied to computers with $t_R \gg 2\, t_c$

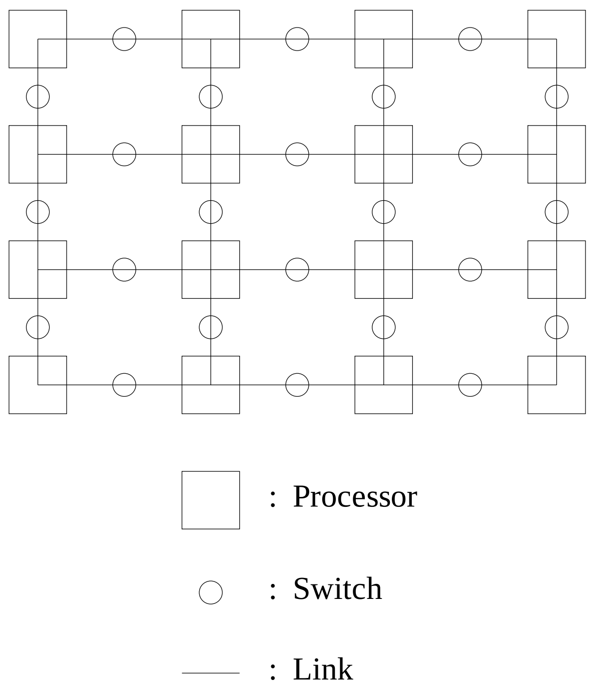


Figure 8 4×4 RMESH

were pointed out in section 4. We also showed how to simulate the mesh algorithms on reconfigurable meshes with buses.

7 References

[BAUD78] G. Baudet, and D. Stevenson, "Optimal Sorting Algorithms for Parallel Computers," *IEEE Transactions on Computers*, C-27, 1, Jan 1978, 84-87.

[JANG92] J. Jang and V. Prasanna, "An optimal sorting algorithm on reconfigurable meshes", International Parallel Processing Symposium, 1992.

[KUMA83] M. Kumar and D.S. Hirschberg, "An efficient implementation of Batcher’s odd-even merge algorithm and its application in parallel sorting schemes," *IEEE Transactions on Computing*, C-32, 3, March 1983, 254-264.

[KNUT73] D. E. Knuth, *The Art of Computer Programming*, Vol 3, *Sorting and*

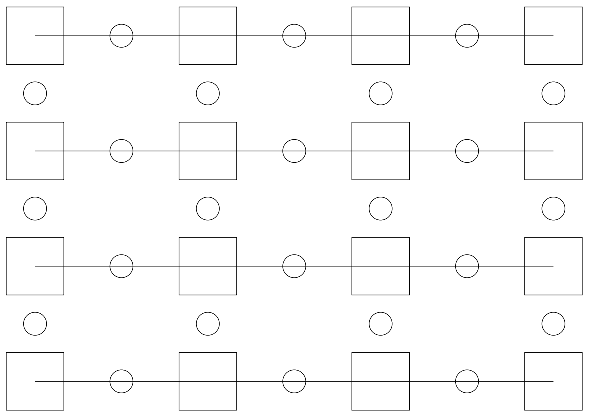


Figure 9 Row buses

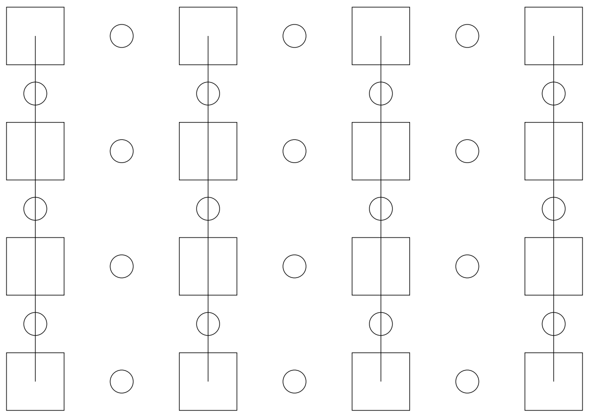


Figure 10 Column buses

Searching, Addison-Wesley, Reading, MA, 1973.

[LEIG85] T. Leighton, "Tight bounds on the complexity of parallel sorting", *IEEE Trans. on Computers*, C-34, 4, April 1985, 344-354.

[LIN92] R. Lin, S. Olariu, J. Schwing, and J. Zhang, "A VLSI-optimal constant

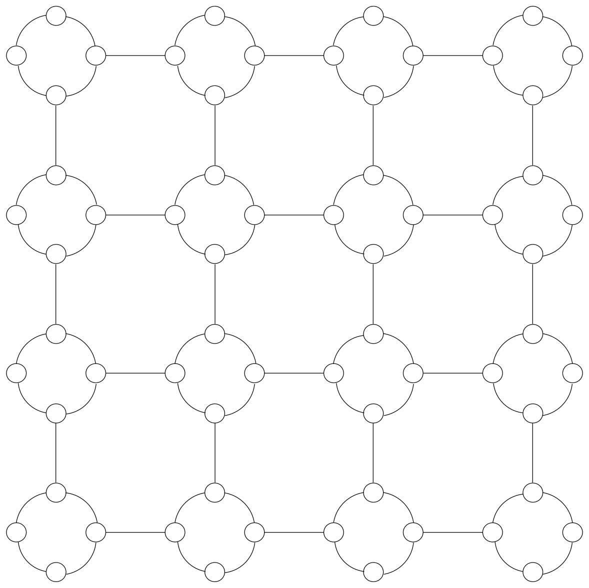


Figure 11 4 x 4 PARBUS

time Sorting on reconfigurable mesh", *Proceedings of Ninth European Workshop on Parallel Computing*, Madrid, Spain, 1992.

[MARB88] John M. Marberg, and Eli Gafni, "Sorting in Constant Number of Row and Column Phases on a Mesh", *Algorithmica*, 3, 1988, 561-572.

[MILL88a] R. Miller, V. K. Prasanna Kumar, D. Reisis and Q. Stout, "Data movement operations and applications on reconfigurable VLS I arrays", *Proceedings of the 1988 International Conference on Parallel Processing*, The Pennsylvania State University Press, 1988, 205-208.

[MILL88b] R. Miller, V. K. Prasanna Kumar, D. Reisis and Q. Stout, "Meshes with reconfigurable buses", Proceedings *5th MIT Conference On Advanced Research In VLSI,* 1988, 163-178.

[MILL88c] R. Miller, V. K. Prasanna Kumar, D. Reisis and Q. Stout, "Image computations on reconfigurable VLSI arrays", *Proceedings IEEE Conference On Computer Vision And Pattern Recognition*, 1988, 925-930.

[MILL91a] R. Miller, V. K. Prasanna Kumar, D. Reisis and Q. Stout, "Efficient parallel algorithms for intermediate level vision analysis on the reconfigurable

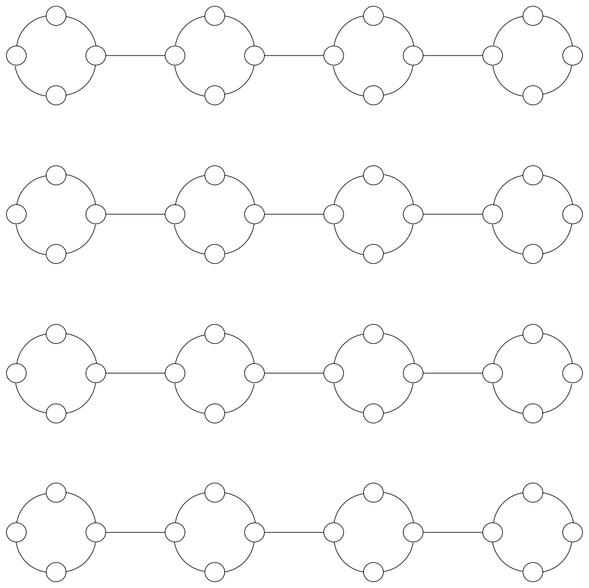


Figure 12 Row buses in a PARBUS

mesh", *Parallel Architectures and Algorithms for Image Understanding*, Viktor K. Prasanna ed., 185-207, Academic Press, 1991

[NASS79] D. Nassimi and S. Sahni, "Bitonic sort on a mesh-connected parallel computer," *IEEE Transactions on Computers*, C-27, 1, Jan 1979, 2-7.

[PARK87] A. Park and K. Balasubramanian, "Improved Sorting Algorithms for Parallel Computers," *Proceedings of 1987 ACM Computer Science Conference*, Feb 1987, 239-244.

[PARK90] A. Park and K. Balasubramanian, "Reducing Communication Costs for Sorting on Mesh-Connected and Linerly Connected Parallel Computers," *Journal of Parallel and Distributed Computing*, 9, 318-322, 1990.

[SCHER89] Issac D. Scherson, Sandeep Sen, and Yiming MA, "Two nearly Optimal Sorting algorithms for Mesh-Connected processor arrays using shear sort," *Journal of Parallel and Distributed Computing*, 6, 151-165, 1989.

[SCHN86] C.P. Schnorr, and A. Shamir, "An Optimal Sorting Algorithm for Mesh

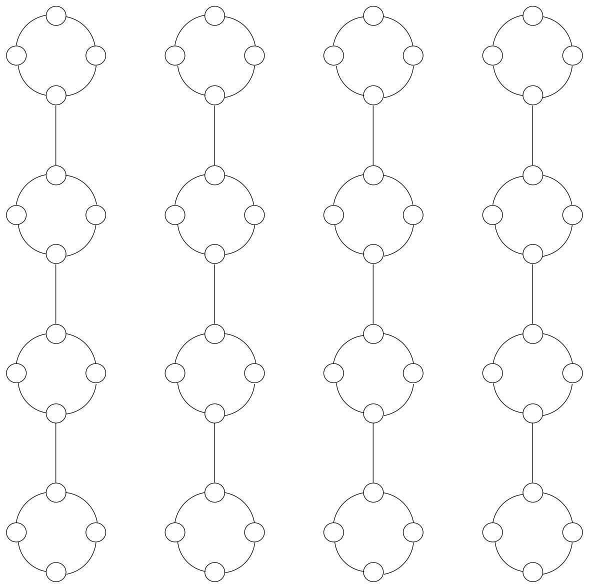


Figure 13 Column buses in a PARBUS

Connected Computers," *Proceeding of the 18th ACM Symposium on Theory of Computing*, May 1985, Berkely, CA, 255-263.

[THOM77] C.D. Thompson, H.T. Kung, "Sorting on a Mesh Connected Parallel Computer," *Communications of ACM*, 20, 4, April 1977, 263-271.

[WANG90a] B. Wang and G. Chen, "Constant time algorithms for the transitive closure and some related graph problems on processor arrays with reconfigurable bus systems," *IEEE Trans. on Parallel and Distributed Systems*, 1, 4, 500-507, 1990.

[WANG90b] B. Wang, G. Chen, and F. Lin, "Constant time sorting on a processor array with a reconfigurable bus system," *Info. Proc. Letrs.*, 34, 4, 187-190, 1990.