Sorting n Numbers On $n \times n$ Reconfigurable Meshes With Buses*

Madhusudan Nigam and Sartaj Sahni

University of Florida

Gainesville, FL 32611

<Revised July 1992>

Technical Report 92-5

ABSTRACT

We show how column sort [LEIG85] and rotate sort [MARB88] can be implemented on the different reconfigurable mesh with buses (RMB) architectures that have been proposed in the literature. On all of these proposed RMB architectures, we are able to sort n numbers on an $n \times n$ configuration in O(1) time.

For the PARBUS RMB architecture [WANG90ab], our column sort and rotate sort implementations are simpler than the O(1) sorting algorithms developed in [JANG92] and [LIN92]. Furthermore, our sorting algorithms use fewer bus broadcasts. For the RMESH RMB architecture [MILL88abc], our algorithms are the first to sort n numbers on an $n \times n$ configuration in O(1) time.

We also observe that rotate sort can be implemented on $N \times N \times \cdots \times N$ $k+1$ dimensional RMB architectures so as to sort N^k elements in O(1) time.

2 **Keywords And Phrases**

Sorting, column sort, rotate sort, reconfigurable mesh with buses.

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

1 Introduction

Several different mesh like architectures with reconfigurable buses have been proposed in the literature. These include the content addressable array processor (CAPP) of Weems et al. [WEEM89], the polymorphic torus of Li and Maresca [LI89, MARE89], the reconfigurable mesh with buses (RMESH) of Miller et al. [MILL88abc], the processor array with a reconfigurable bus system (PARBUS) of Wang and Chen [WANG90], and the reconfigurable network (RN) of Ben-Asher et al. [BENA91].

The CAPP [WEEM89] and RMESH [MILL88abc] architectures appear to be quite similar. So, we shall describe the RMESH only. In this, we have a bus grid with an $n \times n$ arrangement of processors at the grid points (see Figure 1 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 2). Column buses are obtained by having each processor disconnect the switch on its east (Figure 3). 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 4) [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}, $1 \leq i \leq 2$ 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 5 and 6). 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}.

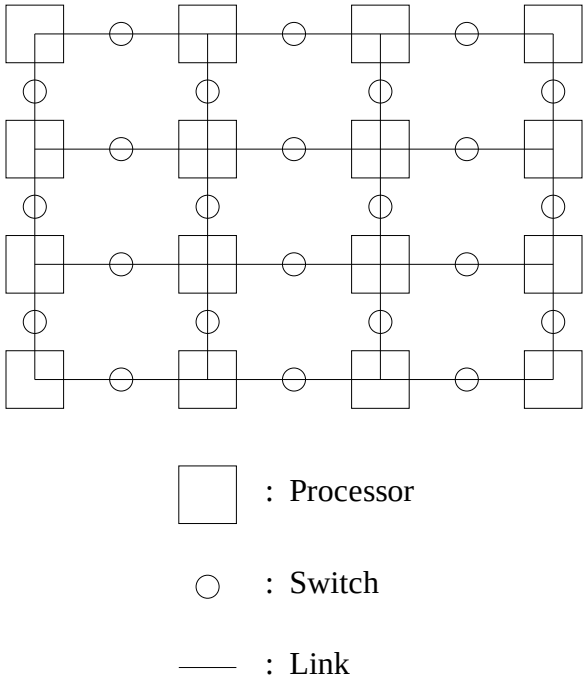


Figure 1 4×4 RMESH

The polymorphic torus architecture [LI89ab, MARE89] is identical to the PARBUS except that the rows and columns of the underlying mesh wrap around (Figure 7). In a reconfigurable network (RN) [BENA91] no restriction is placed on the bus segments that connect pairs of processors or on the relative placement of the processors. I.e., processors may not lie at grid points and a bus segment may join an arbitrary pair of processors. Like the PARBUS and polymorphic torus, each processor has an internal switch that is able to connect together arbitrary subsets of the bus segments that connect to the

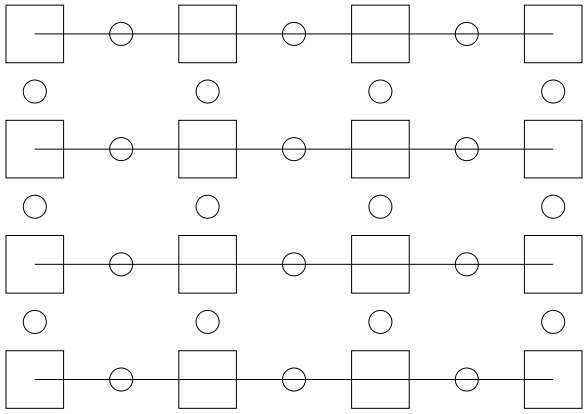


Figure 2 Row buses

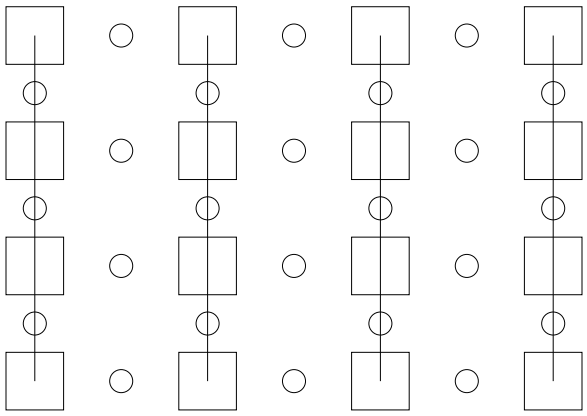


Figure 3 Column buses

processor. Ben-asher et al. [BENA91] also define a mesh restriction (MRN) of their reconfigurable network . In this, the processor and bus segment arrangement is exactly as for the PARBUS (Figure 4). However the switches internal to processors are able to

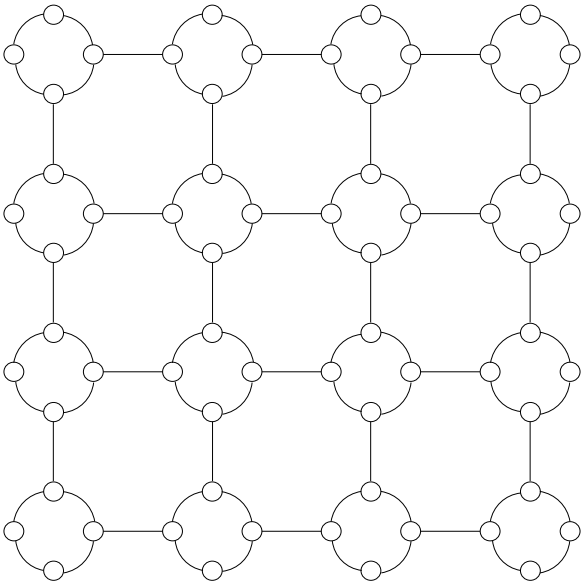


Figure 4 4 x 4 PARBUS

obtain only the 10 bus configurations given in Figure 8. Thus an MRN is a restricted PARBUS.

While we have defined the above reconfigurable bus architectures as square two dimensional meshes, it is easy to see how these may be extended to obtain non square architectures and architectures with more dimensions than two.

In this paper we consider the problem of sorting n numbers on an RMESH, PARBUS and MRN. This sorting problem has been previously studied for all three architectures. n numbers can be sorted in O(1) on a three dimensional $n \times n \times n$ RMESH [JENQ91ab], PARBUS [WANG90], and MRN [BENA91]. All of these algorithms are based on a count sort [HORO90] and are easily modified to run in the same amount of time on a two dimensional $n^2 \times n$ computer of the same model. Nakano et al. [NAKA90] have shown how to sort n numbers in O(1) time on an ($n \log^2 n \times n$) PARBUS. Jang and Prasanna [JANG92] and LIN et al. [LIN92] have reduced the number of processors required by an O(1) sort further. They both present O(1) sorting algorithms that work on an $n \times n$ PARBUS. Since such a PARBUS can be realized using n^2 area, their algorithms

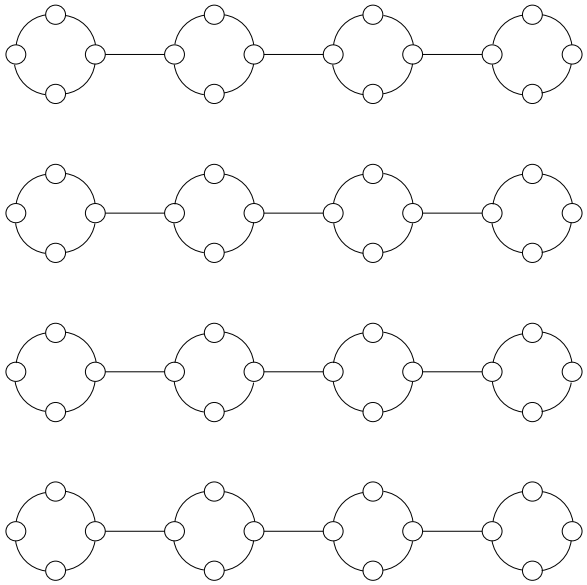


Figure 5 Row buses in a PARBUS

achieve the area time squared (AT^2) lower bound of Ω (n^2) for sorting n numbers in the VLSI word model [LEIG85]. The algorithm of Jang and Prasanna [JANG92] is based on Leighton's column sort [LEIG85] while that of LIN et al. [LIN92] is based on selection. Neither is directly adaptable to run on an $n \times n$ RMESH in O(1) time as the algorithm of [JANG92] requires processors be able to connect their bus segments according to A = { {N,S}, {E,W}} while the algorithm of [LIN92] requires A = {{N,S}, {E,W}} and {{N,W}, {S, E}} . These are not permissible in an RMESH. Their algorithms are, however, directly usable on an $n \times n$ MRN as the bus connections used are permissible connections for an MRN. Ben-Asher et al. [BENA91] describe an O(1) algorithm to sort n numbers on an RN with $O(n^{1+\epsilon})$ processors for any, $\in$, $\in > 0$. This algorithm is also based on Leighton's column sort [LEIG85].

In this paper, we show how Leighton's column sort algorithm [LEIG85] and Marberg and Gafni's rotate sort algorithm [MARB88] can be implemented on all three

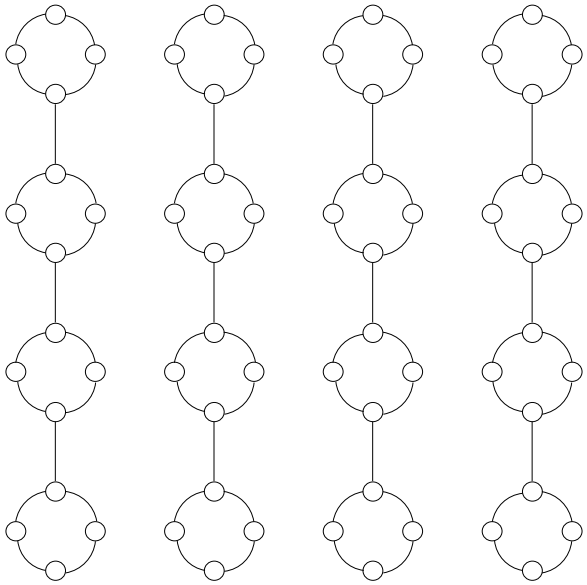


Figure 6 Column buses in a PARBUS

reconfigurble mesh with buses (RMB) architectures so as to sort n numbers in O(1) time on an $n \times n$ configuration. The resulting RMB sort algorithms are conceptually simpler than the O(1) PARBUS sorting algorithms of [JANG92] and [LIN92]. In addition, our implementations use fewer bus broadcasts than do the algorithms of [JANG92] and [LIN92]. Since the PARBUS implementations use only bus connections permissible in an MRN, our PARBUS algorithms may be directly used on an MRN. For an RMESH, our implementations are the first RMESH algorithms to sort n numbers in O(1) time on an $n \times n$ configuation.

In section 2, we describe Leighton's column sort algorithm. Its implementation on an RMESH is developed in section 3. In section 4, we show how to implement column sort on a PARBUS. Rotate sort is considered in sections 5 through 7. In section 5 we describe Marberg and Gafni's rotate sort [MARB88]. The implementation of rotate sort is obtained in sections 6 and 7 for RMESH and PARBUS architectures, respectively. In section 8, we propose a sorting algorithm that is a combination of rotate sort and Scherson et al.'s [SCHER89] iterative shear sort. In section 9, we provide a comparision of the

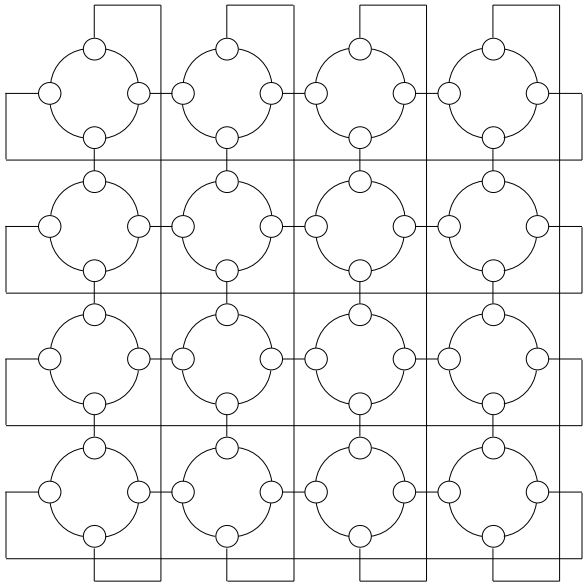


Figure 7 4 x 4 Polymorphic torus

two PARBUS sorting algorithms developed here and those of Jang and Prasanna [JANG92] and Lin et al. [LIN92]. For the PARBUS model, Leighton's column sort uses the fewest bus broadcasts. However, for the RMESH model, combined sort uses the fewest bus broadcasts. In section 10, we make the observation that using rotate sort, one can sort N^k elements, in O(1) time on an $N \times N \times \cdots \times N$ $k+1$ dimensional RMESH and PARBUS.

2 Column Sort

Column sort is a generalization of Batcher's odd-even merge [KNUT73] and was proposed by Leighton [LEIG85]. It may be used to sort an $r \times s$ matrix Q where $r \geq 2(s-1)^2$ and $r \bmod s = 0$. The number of elements in Q is $n = rs$ and the sorted sequence is stored in column major order (Figure 9). Our presentation of column sort follows that of [LEIG85] very closely.

There are eight steps to column sort. In the odd steps 1,3,5, and 7, we sort each

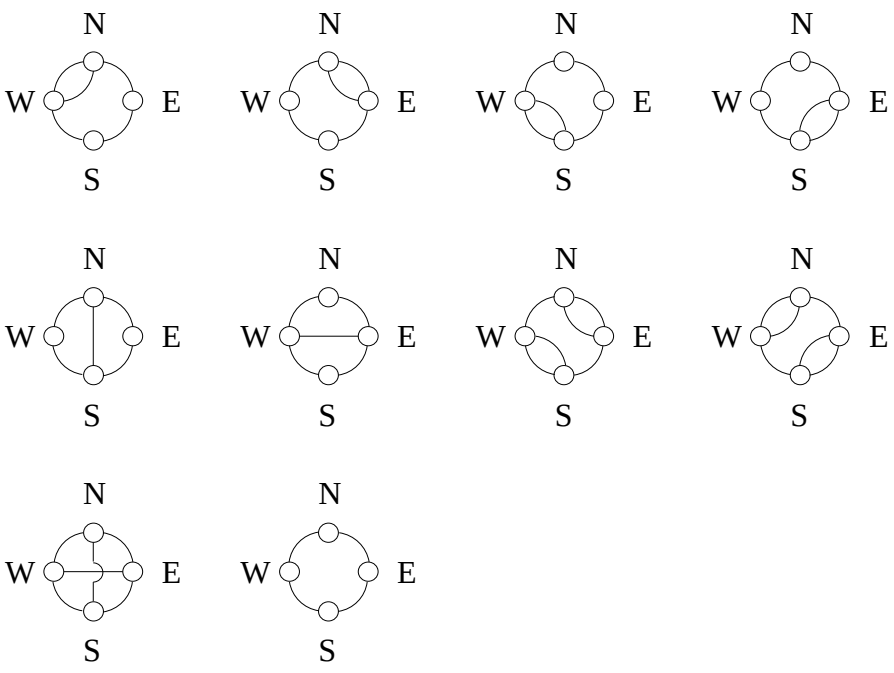
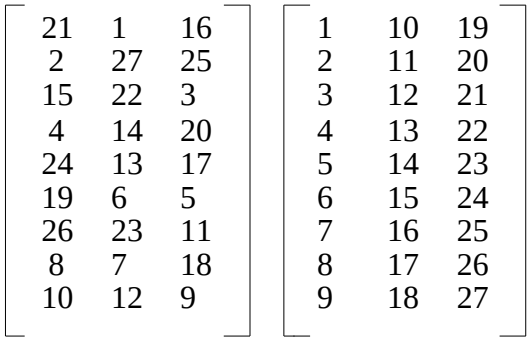


Figure 8 Local Configurations allowed for a switch in MRN

column of Q top to bottom. In step 2, the elements of Q are picked up in column major order and placed back in row major order (Figure 10). This operation is called a *transpose*. Step 4 is the reverse of this (i.e., elements of Q are picked up in row major order and put back in column major order) and is called *untranspose*. Step 6 is a *shift* by $\lfloor \frac{r}{2} \rfloor$. This increases the number of columns by 1 and is shown in Figure 11. Step 8, *unshift*, is the reverse of this. Leighton [LEIG85] has shown that these eight steps are sufficient to sort Q whenever $r \geq 2(s-1)^2$ and $r \bmod s = 0$.



(a) Input Q (b) Output Q

Figure 9 Sorting a 9×3 matrix Q into column major order

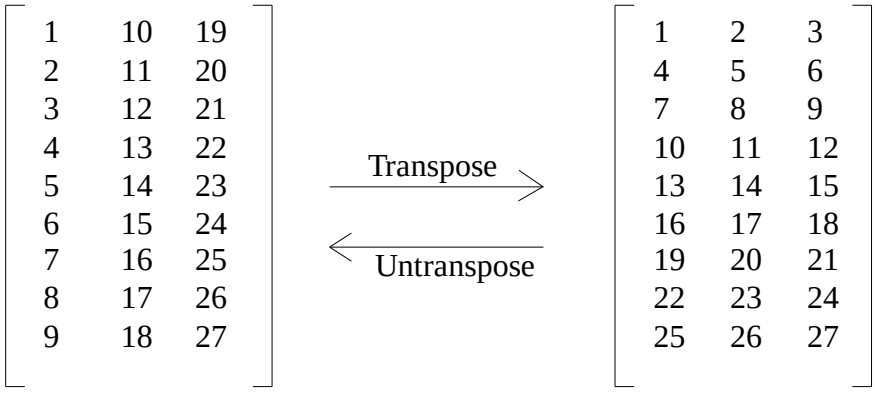


Figure 10 Transpose (step 2) and Untrnnspose (step 4) of a 9×3 matrix

3 Column Sort On An RMESH

Our adaptation of column sort to an $n \times n$ RMESH is similar to the adaptation used by Ben-Asher et al. to obtain an $O(n^{17/9})$ processor reconfigurable network that can sort in

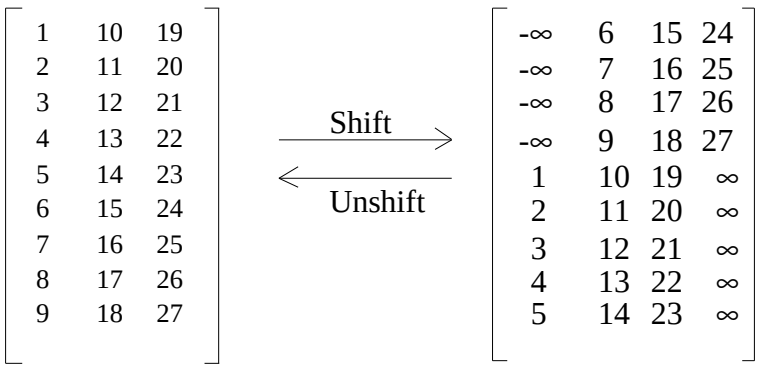


Figure 11 Shift (step 6) and Unshift (step 8)

O(1) time. This reconfigurable network uses approximately 11 layers.

The input n numbers to be sorted reside in the Z variables of the row 1 PEs of an $n \times n$ RMESH. This is interpreted as representing the column major order of the Q matrix used in a column sort (Figure 12). The dimensions of Q are $r_1 \times s_1$ where $r_1 = 1/2*n^{3/4}$ and $s_1 = 2*n^{1/4}$. Clearly, there is an n_1 such that $r_1 \geq 2*(s_1-1)^2$ for all $n \geq n_1$. Hence, column sort will work for $n \geq n_1$. For $n < n_1$ we can sort in constant time (as n_1 is a constant) using any previously known RMESH sorting algorithm.

The steps in the sorting alogrithm are given in Figure 13. Steps 1, 2, and 3 use the $n \times r_1$ sub RMESH A_i to sort column i of Q. For the sort of step 4, the Q matrix has dimensions $r_1 \times (s_1+1)$. Columns 1 and $s_1+ 1$ are already sorted as the -∞'s and +∞'s of Figure 11 do not affect the sorted order. Only the inner s_1 - 1 columns need to be sorted. Each of these columns is sorted using an $n \times r_1$ sub RMESH, B_i, as shown in Figure 14. The sub RMESHs X and Y are idle during this sort.

Thus the sorts of each of the above four steps involve sorting r_1 numbers using an $n \times r_1$ sub RMESH. Each of these is done using column sort with the Q matrix now being an $r_2 \times s_2$ matrix with $r_2 s_2 = r_1$. We use $r_2 = n^{1/2}$ and $s_2 = 1/2*n^{1/4}$. With this choice, we have $2(s_2-1)^2 < 2s_2^2 = 1/2*n^{1/2} < n^{1/2}$, $n \geq 1$. We use W to denote the $r_2 \times s_2$ matrix.

Let us examine step 1 of Figure 13. To sort a column of Q we initially use only the $n \times r_1$ sub RMESH A_i that contains this column. This column is actually the column major

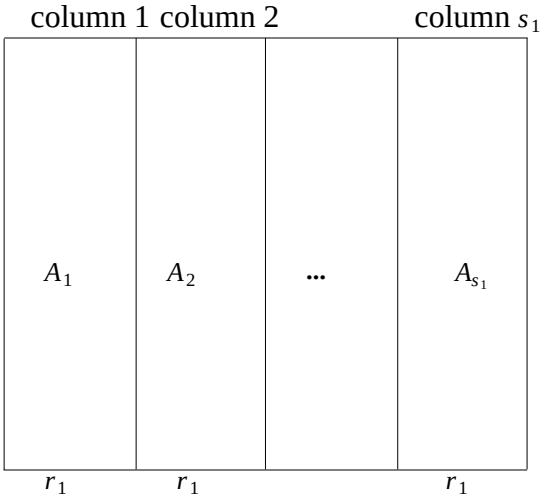


Figure 12 Column major interpretation of Q

representation of a matrix W_i. So, sorting the column is equivalent to sorting W_i. To sort W_i, we follow the steps given in Figure 13 except that Q is replaced by W_i. The steps of Figure 13 are done differently on W_i than on Q. Figure 15 gives the necessary steps for each W_i. Note that this figure assumes that the $n \times n$ RMESH has been partitioned into the s_1 A_i's and each A_i operates independent of each other. To SortTranspose a W_i, we first broadcast W_i which is initially stored in the Z variables of the row 1 PEs of A_i to all rows of A_i (step 1). Following this, the U variable of each PE in each column of A_i is the same. In steps 2-4, each A_i is partitioned into square sub RMESHs B_{ijk}, $1 \le j \le r_2$, $1 \le k \le s_2$ of size $r_2 \times r_2$ (Figure 16). Note that B_{ijk} contains column k of W_i in the U variables of each of its rows. B_{ijk} will be used to determine the rank of j'th element of the k'th column of W_i. Here, by rank we mean the number of elements in the k'th column of W_i that are either less than the j'th element or are equal to it but not in a column to the right of the j'th column of B_{ijk}. So, this rank gives the position, in column k, of the j'th element following a sort of column k. To determine this rank, in step 2, we broadcast the j'th element of column k of W_i to all processors in B_{ijk}. This is done by broadcasting U values from column j of B_{ijk} using row buses that are local to B_{ijk}. The broadcast value is stored in the variables of the

Step 0: [Input] Q is available in column major order, one element per PE, in row 1 of the RMESH. I.e., $z[1,j] = j$'th element of Q in column major order.

Step 1: [Sort Transpose] Obtain the Q matrix following step 2 of column sort. This matrix is available in column major order in each row of the RMESH.

Step 2: [Sort Untranspose] Obtain the Q matrix following step 4 of column sort. This matrix is available in column major order in each row of the RMESH.

Step 3: [Sort Shift] Obtain the Q matrix following step 6 of column sort. This matrix (excluding the $-\infty$ and $+\infty$ values) is available in column major order in each row of the RMESH.

Step 4: [Sort Unshift] Obtain the sorted result in row 1 of the RMESH.

Figure 13 Sorting Q on an RMESH

PEs of B_{ijk}. Now, the processor in position (a,b) of B_{ijk} has the b'th element of column k of W_i in its U variable and the a'th element of this column in its V variable. Step 3 sets S variables in the processors to 0 or 1 such that the sum of the S's in each row of B_{ijk} gives the rank of the j'th element of column k of W_i. In step 4, the S values in a row are summed to obtain the rank. We shall describe shortly how this is done. The computed rank is stored in variable R of the PE in position $[k,1]$ of B_{ijk}. Note that since $k \le s_2 \le r_2$, $[k,1]$ is actually a position in B_{ijk}.

Now if our objective is simply to sort the columns of W_i, we can route the V variable in PE[k,i] (this is the j'th element in column k of W_i) to the R'th column using a row bus local to B_{ijk} and then broadcast it along a column bus to all PE's on column R. However, we wish to transpose the resulting W_i which is sorted by columns. For this, we see that the j'th element of W_i will be in column k and row R of the sorted W_i. Following the transpose, this element will be in row $(k\text{-}1)r_2/s_2 + \lceil \frac{R}{s_2} \rceil$ and column $(R\text{-}1) \bmod s_2 + 1$ of W_i. Hence,

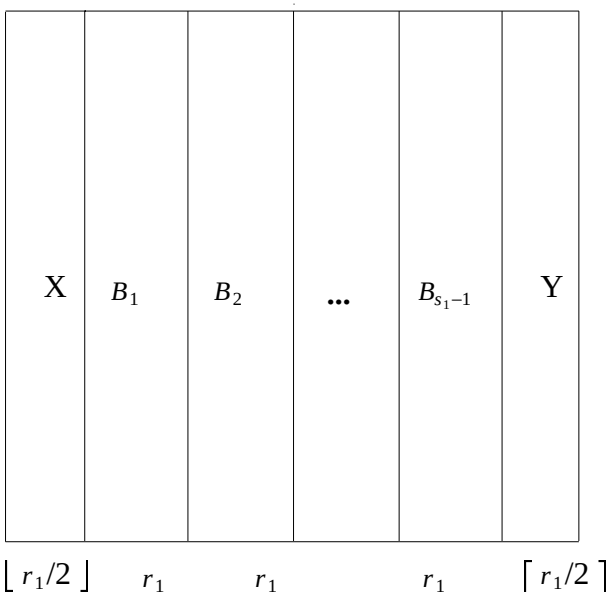


Figure 14 Sub RMESHs for step 4 sort

the V value in PE[k,1] of B_{ijk} is to be in all PEs of column [(R-1) mod s_2]r_2 +(k-1)r_2/s_2 + $\lceil \frac{R}{s_2} \rceil$ of A_i following the SortTranspose. This is accomplished in steps 5 and 6. Note that there are no bus conflicts in the row bus broadcasts of step 5 as the broadcast for each B_{ijk} uses a different row bus that spans the width of A_i. The total number of broadcasts is 4 plus the number needed to sum the S values in a row of B_{ijk}. We shall shortly see that summing can be done with 6 broadcasts. Hence SortTranspose uses 10 broadcasts.

3.1 Ranking

To sum the S values in a row of an $r_2 \times r_2$ RMESH, we use a slightly modified version of the ranking alogrithm of Jenq and Sahni [JENQ91ab]. This algorithm ranks the row 1 processors of an $r_2 \times r_2$ RMESH. The ranking is done by using 0/1 valued variables, S, in row 1. The rank of the processor in column i of row 1 is the number of

Step 1: Broadcast the row 1 Z values, using column buses, to the U variables of all PEs in the same column of A_i.

Step 2: The column j PEs in all B_{ijk}'s broadcast their U values, using row buses local to each B_{ijk}, to the V variables of all PEs in the same B_{ijk}.

Step 3: PE $[a,b]$ of B_{ijk} sets its S value to 1 if ($U < V$) or ($U = V$ and $b < j$). Otherwise, it sets its S value to 0. This is done, in parallel, by all PEs in all B_{ijk}'s.

Step 4: The sum of the S's in any one row of B_{ijk} is computed and stored in the R variable of PE$[k,1]$ of B_{ijk}. This is done. in parallel, for all B_{ijk}'s.

Step 5: Using row buses, PE$[k,1]$ of each B_{ijk} sends its V value to the PE in column $[(R\text{-}1) \bmod s_2]r_2 + (k\text{-}1)r_2/s_2 + \lceil \frac{R}{s_2} \rceil$ of A_i

Step 6: The PEs that received V values in step 5 broadcast this value, using column buses, to the U variables of the PEs in the same column of A_i.

Figure 15 Steps to Sort Transpose W_i into A_i

processors in row 1 and columns k, $k < i$ that have $S = 1$. Hence, the rank of processor $[1,r_2]$ equals the sum of the S values in row 1 except when $S[1,r_2] = 1$. In this latter case we need to add 1 to the rank to get the sum. The ranking algorithm of [JENQ91ab] has three phases associated with it.

The procedures for phases 1 and 2 are quite similar. These are easily modified to start with the configuration following step 3 of Figure 15 and send the phase 1 and phase 2 results of the rightmost odd and even columns in B_{ijk} to PE$[k,1]$ where the two results are added. To avoid an extra broadcast, the result for the rightmost column (phase 1 if r_2 is even and phase 2 if r_2 is odd) is incremented by 1 in case $S[*,r_2] = 1$ before the

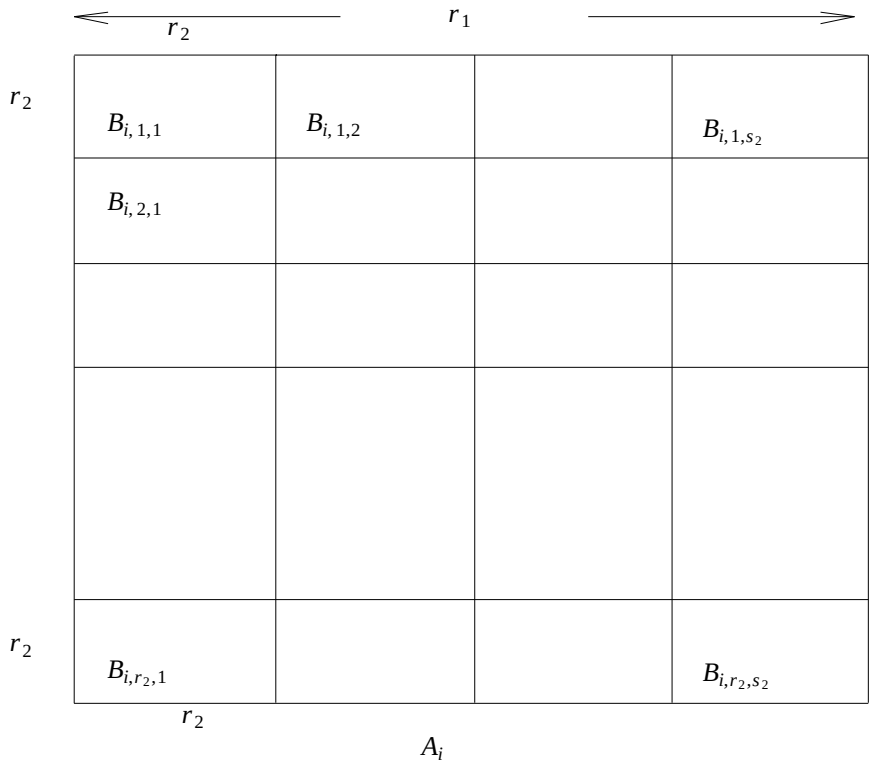


Figure 16 Division of A_i into r_2 x r_2 sub RMESHs

braodcast. While the phase 1 code of [JENQ91ab] uses 4 broadcasts, the first of these can be eliminated as we begin with a configuration in which $S[*,b]$ is already on all rows of column b, $1 \le b \le r_2$. So, phases 1 and 2 use 6 broadcasts. Phase 3 of [JENQ91ab] uses two broadcasts. Both of these can be eliminated by having their phase 1 (2) step 10 directly broadcat the rank of the rightmost even (odd) column to PE[k,1] bus using a row bus that spans row k connected to a column bus that spans the rightmost even (odd) column of B_{ijk}. So, the summing operation of step 4 can be done using a total of 6 broadcasts.

Phase 1: Rank the processors in even columns of row 1. This does not account for the 1's in the odd column.

Phase 2: Phase 2: Rank the processors in odd columns of row 1. This does not account for 1's in the even columns.

Phase 3: Combine odd and even ranks.

3.2 Analysis of RMESH Column Sort

First, let us consider sorting a column of Q which is an $r_1 \times s_1$ matrix. This requires us to perform steps 1-4 of Figure 13 on the W_i's. As we have just seen, step 1 uses 10 broadcasts. Step 2 is similar to step 1 except that we begin with the data on all rows of A_i and instead of a transpose, an untranspose is to performed. This means that step 1 of Figure 15 can be eliminated and the formula in step 5 is to be changed to correspond to an untranspose. The number of broadcasts for step 2 of Figure 13 is therefore 9. Steps 3 and 4 are similar and each uses 9 broadcasts. The total number of broadcasts to sort a column of Q is therefore 37.

Now, to perform a SortTranspose of the columns of Q we proceed as in a sort of the columns of Q except that the last broadcast of SortUnshift performs the transpose and leaves the transposed matrix in column major order in all rows of the RMESH. This takes 37 broadcasts. The SortUntranspose takes 36 broadcasts as it begins with Q in all rows. Similarly, step 3 and 4 each take 36 broadcasts. The total number of broadcats is therefore 145.

A more careful analysis reveals that the number of broadcasts needed for the SortShift and SortUnshift steps can be reduced by one each as the step 5 (Figure 15) broadcast can be eliminated. Taking this into account, the number of broadcasts to sort a column of Q becomes 35. However to do a SortTranspose of the columns of Q, we need to do an additional broadcast during the SortUnshift of the W_i's. This brings the number to 36. A SortUntranspose of Q takes 35 broadcasts and the remaining two steps of Figure

10 each takes 34 broadcasts. Hence, the total number of broadcasts becomes 139.

4 Column Sort On A PARBUS

The RMESH algorithm of Section 3 will work on a PARBUS as all connections used by it are possible in a PARBUS. We can, however, sort using fewer than 139 broadcasts on a PARBUS. If we replace the ranking algorithm of [JENQ91ab] that we used in Section 3 by the prefix sum of [LIN92] (i.e., prefix sum N bits on a $(N+1)\times N$ PARBUS) then we can sum the S's in a row using two broadcasts. The algorithm of [LIN92] needs to be modified slightly to allow for the fact that we begin with the S values on all columns and we are summing r_2 S values on a $r_2 \times r_2$ PARBUS rather than an $(r_2+1)\times r_2$ PARBUS. These modifications are straightforward. Since the S's can be summed in 2 broadcasts rather than 6, the SortTranspose of Figure 13 requires only 6 broadcasts. The SortUntranspose canbe done in 5 broadcasts. As indicated in the analysis of Section 3, the SortShift and SortUnshift steps can be done without the broadcast of Step 5 of Figure 15. So, each of these require only 4 broadcasts. Thus, to sort a column of Q requires 19 broadcasts. Following the analysis of Section 3, we see that a SortTranspose of Q can be done with 20 broadcasts, a SortUntranspose with 19 broadcasts, and a SortShift and SortUnshift with 18 broadcasts each. Thus n numbers can be sorted on an $n \times n$ PARBUS using 75 broadcasts.

We can actually reduce the number of broadcasts further by begining with $r_1 = n^{2/3}$ and $s_1 = n^{1/3}$. While this does not satisfy the requirement that $r \geq 2(s-1)^2$, Leighton [LEIG85] has shown that column sort works for r and s such that $r \geq s(s-1)$ provided that the Untranspose of step 4 is replaced by an Undiagonalize step (Figure 17). The use of the undiagonalizing permutation only requires us to change the formula used in step 5 of Figure 15 to a slightly more complex one. This does not change the number of broadcasts in the case of the RMESH and PARBUS algorithms previously discussed. However, the ability to use $r_1 = n^{2/3}$ and $s_1 = n^{1/3}$ (instead of $r_1 = 2\ n^{2/3}$ and $s_1 = 1/2{*}n^{1/3}$ which satisfy $r \geq 2(s-1)^2$) significantly reduces the number of broadcasts for the PARBUS algorithm we are about to describe.

Now, the SortTranspose, SortUntranspose, SortShift and SortUnshift for Q are not done by using another level of column sort. Instead a count sort similar to that of Figure 15 is directly applied to the columns of Q. This time, A_i is an $n \times r_1 = n \times n^{2/3}$ sub PARBUS. Let D_{ij} be the j'th (from the top) $n^{1/3} \times r_1$ sub PARBUS of A_i (Figure 18). The D_{ij}'s are used to do the counting previously done by the B_{ijk}'s. To count $r_1 = n^{2/3}$ bits using an $n^{1/3} \times n^{2/3}$ sub PARBUS, we use the parallel prefix sum algorithm of [LIN92] which

does this in 12 broadcasts when we begin with bits in all rows of D_{ij} and take into account we want only the sum and not the prefix sum. Note that the prefix sum algorithm of [LIN92] is an iterative algorithm that uses modulo M arithmetic to sum N bits on an $(M+1)\times N$ PARBUS. For this it uses $\frac{\log N}{\log M}$ iterations. For the case of summing N bits, this is easily modified to run on an $M\times N$ PARBUS in $\frac{\log N}{\log M}$ iterations with each iteration using 6 broadcasts (3 for the odd bits, and 3 for the even bits). With our choice of r_1 and s_1, the number of iterations is 2, while with $r_1 = 2\ n^{2/3}$ and $s_1 = 1/2\ n^{1/3}$, the number of iterations is 3. The two iterations, together, use 12 broadcasts.

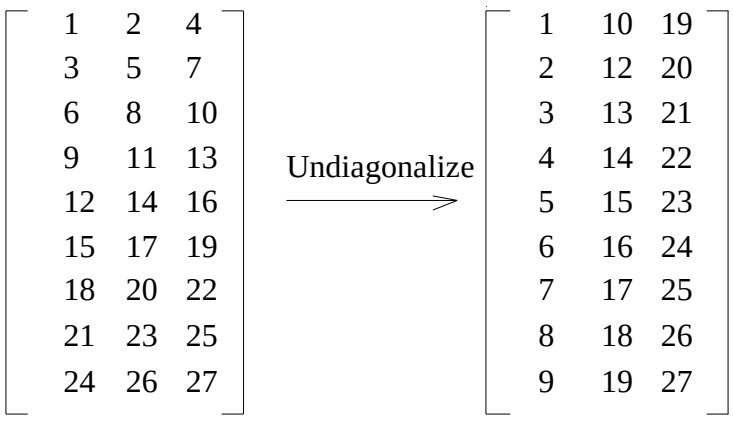


Figure 17 Undiagonalize a 9×3 matrix

To get the SortTranspose algorithm for the PARBUS, we replace all occurrences of B_{ijk} by D_{ij} and of PE[k,1] by PE[i,1] in Figure 15. The formula of step 5 is changed to [(R-1) mod s_1] r_1 + (i-1)r_1/s_1 + $\lceil \frac{R}{s_1} \rceil$. The number of broadcasts used by the new SortTranspose algorithm is 16. The remaining three steps of Figure 15 are similar. The SortUndiagonalize takes 15 broadcasts as it begins with data in all rows and the step 1 (Figure 11) broadcast can be eliminated. The SortShift and SortUnshift each can be done in 14 broadcasts as the step 5 (Figure 15) broadcasts are unnecessary. So, the number of broadcasts in the one level PARBUS column sort algorithms is 59.

$$A_i = n \times r_1 = n \times 2n^{2/3}$$

Figure 18 Decomposing A_i into D_{ij}'s

5 Rotate Sort

Rotate sort was developed by Marberg and Gafni [MARB88] to sort $M\,N$ numbers on an $M \times N$ mesh with the standard four neighbor connections. To state their algorithm we need to restate some of the definitions from [MARB88]. Assume that $M = 2^s$ and $N = 2^{2t}$ where $s \geq t$. An $M \times N$ mesh can be tiled in a natural way with tiles of size $M \times N^{1/2}$. This tiling partitions the mesh into vertical slices (Figure 19(a)). Similarly an $M \times N$ mesh can be tiled with $N^{1/2} \times N$ tiles to obtain horizontal slices (Figure 19(b)). Tiling by $N^{1/2} \times N^{1/2}$ tiles results in a partitioning into blocks (Figure 19(c)). Marberg and Gafni define three procedures on which rotate sort is based. These are given in Figure 20.

Rotate sort is comprised of the six steps given in Figure 21. Recall that a vertical slice is an $M \times N^{1/2}$ submesh; a horizontal slice is an $N \times N^{1/2}$ submesh; and a block is a $N^{1/2} \times N^{1/2}$ submesh.

Marberg and Gafni [MARB88] point out that when $M = N$, step 1 of rotate sort may be replaced by the steps.

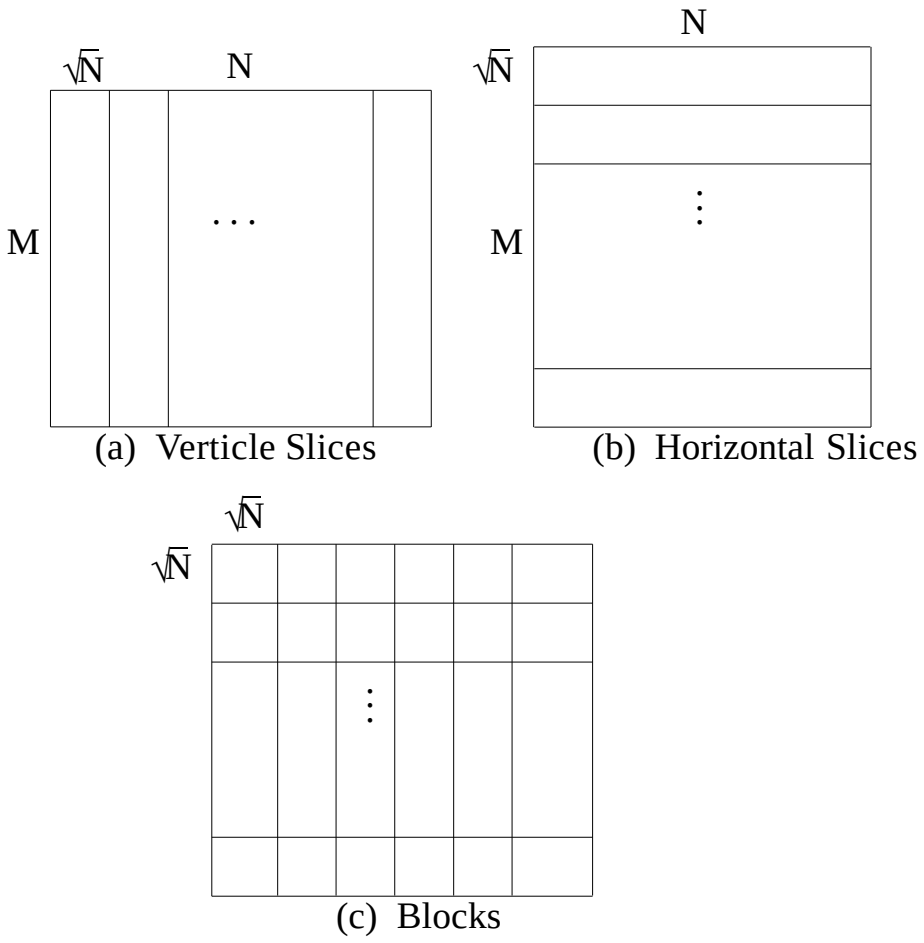


Figure 19 Definitions of the Slice and block

Step 1' (a) Sort all the columns downward;

(b) Sort all the rows to the right;

This simplifies the algorithm when it is implemented on a mesh. However, it does not reduce the number of bus broadcasts needed on reconfigurable meshes with buses. As a result we do not consider this variant of rotate sort.

Procedure balance (v,w);

{Operate on submeshes of size $v \times w$}

sort all columns of the submesh downward;

rotate each row i of the submesh, i mod w positions right;

sort all columns of the submesh downward;

end;

(a) Balance a submesh

Procedure unblock;

{Operate on entire mesh}

rotate each row i of the mesh $(i.N^{1/2})$ mod N positions right;

sort all columns of the block downwards;

end;

(b) Unblock

Procedure shear;

{Operate on entire mesh}

sort all even numbered rows to the right and all odd numbered rows to the left;

sort all columns downward;

end;

(c) Shear

Figure 20 Procedures from [MARB88]

6 Rotate Sort On An RMESH

In this section, we adapt the rotate sort algorithm of Figure 21 to sort n numbers on an $n \times n$ RMESH. Note that the algorithm of Figure 21 sorts MN numbers on an $M \times N$ mesh. For the adaptation, we use $M = N$. Hence, $n = N^2 = 2^{4t}$. The $n = N^2$ numbers to be sorted are available, initially, in the Z variable of the row 1 PEs of the $n \times n$ RMESH. We assume a row major mapping from the $N \times N$ mesh to row 1 of the RMESH. Figure 22

Step 1: *balance* each vertical slice;

Step 2: *unblock* the mesh;

Step 3: *balance* each horizontal slice as if it were an $N \times N^{1/2}$ mesh lying on its side;

Step 4: *unblock* the mesh;

Step 5: *shear* three times;

Step 6: *sort* rows to the right;

Figure 21 Rotate Sort [MARB88]

gives the steps involved in sorting the columns of the $n \times n$ mesh downward. Note that this is the first step of the balance operation of step 1 of rotate sort. The basic strategy employed in Figure 22 is to use each $n \times N$ subRMESH of the $n \times n$ RMESH to sort one column of the $N \times N$ mesh.

For this, we need to first extract the columns from row 1 of the RMESH. This is done in steps 1-3 of Figure 22. Following this, each row of the q'th $n \times N$ subRMESH contains the q'th column of the $N \times N$ mesh. Steps 4-6 implement the count phase of a count sort. This implementation is equivalent to that used in [JENQ91b] to sort m elements on an $m \times m \times m$ RMESH. Steps 7 and 8 route the data back to row 1 of the RMESH so that the Z values in row 1 (and actually in all rows) correspond to the row major order of the $n \times n$ mesh following a sort of its columns. The total number of broadcasts used is 12 (note that step 6 uses 6 broadcasts).

The row rotation required by procedure *balance* can be obtained at no aditional cost by changing the destination column computed in step 7 of Figure 22 so as to account for the rotation. The second sort of the columns performed in procedure *balance* can be done with 9 additional broadcasts. For this, during the first column sort of procedure *balance*, the step 7 broadcast of Figure 22 takes into account both the row rotation and the row major to column major transformation to be done in steps 1-3 of Figure 22 for the second column sort of procedure *balance*. So, step 1 of rotate sort (i.e., balancing the vertcial slices) can be done using a total of 21 broadcasts.

To unblock the data, we need to rotate each row and then sort the columns downward. Once again, the rotation can be accomplished at no additional cost by modifying

{sort columns of $N \times N$ mesh}

Step 1: Use column buses to broadcast $Z[1,j]$ to all rows in column j, $1 \le i \le n$. Now, $Z[i,j] = Z[1,j]$, $1 \le i \le n$, $1 \le j \le n$.

Step 2: Use row buses to broadcast $Z[i,i]$ to the R variable of all PEs on row i of the RMESH. Now, $R[i,j] = Z[i,i] = Z[1,i]$, $1 \le i \le n$, $1 \le j \le n$.

Step 3: In the q'th $n \times N$ subRMESH all, PEs$[i,j]$ such that $i \bmod N = q \bmod N$ and $(j-1) \bmod N + 1 = \lceil i/N \rceil$ broadcast their R values along the column buses, $1 \le q \le N$. Note that each such PE$[i,j]$ contains the $\lceil i/N \rceil$'th element of column q of the $N \times N$ mesh. This value is broadcast to the U variables of the PEs in the column. Now, each row of the q'th $n \times N$ subRMESH contains in its U variables column q of the $n \times n$ mesh.

Step 4: Now assume that the $n \times n$ RMESH is tiled by N^2 $N \times N$ subRMESHs. In the $[a,b]$'th such subRMESH, the PEs on column a of the subRMESH broadcast their U value using row buses local to the subRMESH. This is stored in the V variable of each PE.

Step 5: PE $[i,j]$ of the $[a,b]$'th subRMESH sets its S value to 0 if $(U < V)$ or $(U = V$ and $i < a)$.

Step 6: The sum of the S's in any one row of each of the $N \times N$ subRMESH's is computed. The result for the $[a,b]$'th subRMESH is stored in the T variable of PE $[b, 1]$.

Step 7: Using the row buses that span the $n \times n$ RMESH the PE in position $[b,1]$ of each $N \times N$ subRMESH $[a,b]$ sends its V value to the Z variable of the PE in column $(b-1)N + T + 1$.

Step 8: The received Z values are broadcast along column buses.

Figure 22 Sorting the columns of an $N \times N$ mesh

the destination column function used in step 7 of the second column sort performed during the vertical slice balancing of step 1. The column sort needs 11 broadcasts.

The horizontal slices can be balanced using 18 broadcasts as the Z data is already

distributed over the columns. The unblock of step 4 takes as many broadcasts as the unblock of step 2 (i.e. 9).

The shear opertaion requires a row sort followed by a column sort. Row sorts are performed using the same strategy as used for a column sort. The fact that all elements of a row are in adjacent columns of the RMESH permits us to eliminate steps 1-3 of Figure 22. So, a row sort takes only 9 additional broadcasts. The following column sort uses 9 broadcasts. So, each application of shear takes 18 broadcasts. Since we need to shear three times, step 5 of rotate sort uses 54 broadcasts. Step 6 of rotate sort is a row sort. This takes 9 broadcasts. The total number of broadcasts is $21 + 9 + 18 + 9 + 54 + 9 = 120$. This is 19 fewer than the number of broadcasts used by our RMESH implementation of column sort.

7 Rotate Sort On A PARBUS

Our implementation of rotate sort on a PARBUS is the same as that on an RMESH. Note, however, that on a PARBUS ranking (step 6 of Figure 22) takes only 3 broadcasts. Since this is done once for each/row column sort and since a total of 13 such sorts is done, 39 fewer broadcasts are needed on a PARBUS. Hence our PARBUS implementation of rotate sort takes 81 broadcasts. Recall that Leighton's column sort could be implemented on a PARBUS using only 59 broadcasts.

8 A Combined Sort

We may combine the first three steps of the iterative shear sort algorithm of Scherson et al. [SCHER89] with the last four steps of rotate sort to obtain combined sort of Figure 23. This is stated for an $N \times N$ mesh using nearest neighbor connections. The number of elements to sorted is N^2.

Notice that step 4-7 of Figure 23 differ from steps 3-6 of Figure 21 only in that in step 6 of Figure 23 the shear sort is done two times. The correctness of Figure 23 may be established using the results of [SCHER89] and [MARB88].

To implement the combined sort on an $n \times n$ RMESH or $n \times n$ PARBUS ($n = N^2$ elements to be sorted), we note that the column sort of step 3 can be done in the same manner as the column sorts of rotate sort are done. The shift of step 2 can be combined with the sort of step 1. The block sort of step 1 is done using submeshes of size $n \times n^{3/4} = N^2 \times N^{3/2} = N^2 \times (N^{3/4} \times N^{3/4})$. On a PARBUS, this is done by ranking in $n^{1/4} \times n^{3/4}$

Step 1: Sort each $N^{3/4} \times N^{3/4}$ block;

Step 2: Shift the i'th row by $(i*N^{3/4}) \bmod N$ to the right, $1 \le i \le N$;

Step 3: Sort the columns downward;

step 4: *balance* each horizontal slice as if it were an $N \times N^{1/2}$ mesh lying on its side;

Step 5: *unblock* the mesh;

Step 6: *shear* two times;

Step 7: *sort* rows to the right;

Figure 23 Combined Sort

as in [LIN92] while on an RMESH, this is done using the algorithm to sort a column of Q using an $n \times r_1$ submesh (section 3). We omit the details here. The number of broadcasts is 77 for PARBUS and 118 for an RMESH.

9 Comparison With Other O(1) PARBUS Sorts

As noted earlier, our PARBUS implementation of Leighton's column sort uses only 59 broadcasts whereas our PARBUS implementation of rotate sort uses 81 broadcasts and our implementation of combined sort uses 77 broadcasts. The O(1) PARBUS sorting algorithm of Jang and Prasanna [JANG92] is also based on column sort. However, it is far more complex than our adaptation and uses more broadcasts than does the O(1) PARBUS algorithm of Lin et al. [LIN92]. So, we compare our algorithm to that of [LIN92]. This latter algorithm is not based on column sort. Rather, it is based on a multiple selection algorithm that the authors develop. This multiple selection algorithm is itself a simple modification of a selection algorithm for the PARBUS. This algorithm selects the k'th largest element of an unordered set S of n elements. The multiple selection algorithm takes as input an increasing sequence $q_1 < q_2 < ... < q_{n^{1/3}}$ with $1 \le q_i \le n$ and reports for each i, the q_i'th largest element of S. By selecting $q_i = i*n^{2/3}$, $1 \le i \le n^{1/3}$ one is able to determine partitioning elements such that the set of n numbers to be sorted can be partitioned into $n^{1/3}$ buckets each having $n^{2/3}$ elements. Each bucket is then sorted using an $n \times n^{2/3}$ sub PARBUS. Lin et al. [LIN92] were only concerned with developing a constant time algorithm to sort n numbers on an $n \times n$ PARBUS. Consequently, they did not

attempt to minimize the number of broadcasts needed for a sort. However, we analyzed versions of their algorithms that were optimized by us. The optimized selection algorithm requires 84 broadcasts and the optimized sort algorithm used 103 broadcasts. Thus our PARBUS implementation of Leighton's column sort uses slightly more than half the number of broadcasts used by an optimized version of LIN et al.'s algorithm. Furthermore, even if one were interested only in the selection problem, it would be faster to sort using our PARBUS implementation of Leighton's column sort algorithm and then select the k'th element than to use the optimized version of the PARBUS selection algorithm of [LIN92]. Our algorithm is also conceptually simpler than those of [JANG92] and [LIN92]. Like the algorithms of [JANG92] and [LIN91], our PARBUS algorithms may be run directly on an $n \times n$ MRN. The number of broadcasts remains unchanged.

10 Sorting On An $n^{1/2} \times n^{1/2} \times n^{1/2}$ Reconfigurable Mesh with Buses

Rotate sort works by sorting and/or shifting rows and columns of an $N \times N$ array. This algorithm may be implemented on a three dimensional; $N \times N \times N$ reconfigurable mesh with buses so as to sort N^2 elements in O(1) time. In other words, $n = N \times N$ elements are being sorted in O(1) time on an $n^{1/2} \times n^{1/2} \times n^{1/2}$ RMB. Assume that we start with n elements on the base of an $N \times N \times N$ RMB. To sort row (column) i, we broadcast the row (column) to the i'th layer of $N \times N$ processors and sort it in O(1) time in this layer using the algorithms developed in this paper to sort N elements on an $N \times N$ RMB. The total number of such sorts required by rotate sort is 13 (i.e., O(1)) and the required shifts may be combined with the sorts.

We can extend this to obtain an algorithm to sort N^k numbers on a $k+1$ dimensional RMB with N^{k+1} processors in O(1) time. The RMB has an $N \times N \times \cdots \times N$ configuration and the N^k numbers are initially in the face with $k+1$ dimension equal to zero. In the preceding paragraph we showed how to do this sort when $k = 2$. Suppose we have an algorithm to sort N^{l-1} numbers on an l dimensional N^l processor RMB in O(1) time. We can use this to sort N^l numbers on an N^{l+1} processor RMB in O(1) time by regarding the N^l numbers as forming an $N^{l-1} \times N$ array. In the terminology of Marberg and Gafni [MARB88], we have an $M \times N$ array with $M = N^{l-1} \geq N$. To use rotate sort, we need to be able to sort the columns of this array (which are of size M); sort the rows which are of size N); and perform shifts/rotations on the rows or subrows.

To do the column sort we use l dimensional RMBs. The mth such RMB consists of all processors with index of the type $[i_1,i_2,...,i_{l-1},m,i_{l+1}]$. By assumption, this RMB can sort its N^{l-1} numbers in O(1) time. To sort the rows, we can use two dimensional RMBs. Each such RMB consists of processors that differ only in their last two dimensions (i.e., $[a,b,c,...,i_l,i_{l+1}]$). This sort is done using the O(1) sorting algorithm for two dimensional RMBs. The row shifts and rotates are easily done in O(1) time using the two dimensional RMBs just described (actually most of these can be combined with one of the required sorts).

11 Conclusions

We have developed relatively simple algorithms to sort n numbers on reconfigurable $n \times n$ meshes with buses. For the case of the RMESH, our algorithms are the first to sort in O(1) time. For the PARBUS, our algorithms are simpler than those of [JANG92] and [LIN92]. Our PARBUS column sort algorithm is the fastest of our algorithms for the PARBUS. It uses fewer broadcasts than does the optimized versions of the selection algorithm of [LIN92]. Our PARBUS algorithms can be run on an MRN with no modifications. Since $n \times n$ reconfigurable meshes require n^2 area for their layout. Our algorithms (as well as those of [JANG92] and [LIN92]) have an area time square product AT^2 of n^2 which is the best one can hope for in view of the lower bound result $AT^2 \geq n^2$ for the VLSI word model [LEIG85].

Using two dimensional meshes with buses, we are able to sort n elements in O(1) time using n^2 processors. Using higher dimensional RMB, one can sort n numbers in O(1) time using fewer processors. In general, $n = N^k$ numbers can be sorted in O(1) time using $N^{k+1} = n^{1+1/k}$ processors in a $k+1$ dimensional configuration. While the same result has been shown for a PARBUS [JANG92b], our algorithm applies to an RMESH also.

12 References

[BENA91] Y. Ben-Asher, D. Peleg, R. Ramaswami, and A. Schuster, "The power of reconfiguration," *Journal of Parallel and Distributed Computing*, 13, 139-153, 1991.

[HORO90] E. Horowitz and S. Sahni, *Fundamentals of data structures in Pascal*,

Third Edition, Computer Science Press, Inc., New York, 1990.

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

[JENQ91a] J. Jenq and S. Sahni, "Reconfigurable mesh algorithms for image shrinking, expanding, clustering, and template matching," *Proceedings 5th International Parallel Processing Symposium*, IEEE Computer Society Press, 208-215, 1991.

[JENQ91b] J. Jenq and S. Sahni, "Reconfigurable mesh algorithms for the Hough transform," *Proc. 1991 International Conference on Parallel Processing*, The Pennsylvania State University Press, 34-41, 1991.

[JENQ91c] J. Jenq and S. Sahni, "Reconfigurable mesh algorithms for the area and perimeter of image components," *Proc. 1991 International Conference on Parallel Processing*, The Pennsylvania State University Press, 280-281, 1991.

[KNUT73] D. E. Knuth, *The Art of Computer Programming*, Vol 3, Addison-Wesley, NewYork, 1973.

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

[LI89a] H. Li and M. Maresca, "Polymorphic-torus architecture for computer vision," *IEEE Trans. on Pattern & Machine Intelligence*, 11, 3, 133-143, 1989.

[LI89b] H. Li and M. Maresca, "Polymorphic-torus network", *IEEE Trans. on Computers*, C-38, 9, 1345-1351, 1989.

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

[MARE89] M. Maresca, and H. Li, "Connection autonomy in SIMD computers: A VLSI implementation", *Journal of Parallel and Distributed Computing*, 7, April 1989, 302-320.

[MILL88a] R. Miller, V. K. Prasanna Kumar, D. Reisis and Q. Stout, "Data movement

operations and applications on reconfigurable VLSI arrays", *Proceedings of the 1988 International Conference on Parallel Processing*, The Pennsylvania State University Press, pp 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, pp 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, pp 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 mesh", *Parallel Architectures and Algorithms for Image Understanding*, Viktor k. Prasanna ed., 185-207, Academic Press, 1991

[MILL91b] R. Miller, V. K. Prasanna Kumar, D. Reisis and Q. Stout, "Image processing on reconfigurable meshes", *From Pixels to Features II*, H. Burkhardt ed., Elsevier Science Publishing, 1991.

[NAKA90] Koji Nakano, Thoshimits Msuzawa, Nobuki Tokura, "A fast sorting algorithm on a reconfigurable array", Technical Report, COMP 90-69, 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.

[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.

[WEEM89] C. C. Weems, S. P. Levitan, A. R. Hanson, E. M. Riseman, J. G. Nash, and D. B. Shu, "The image understanding architecture, " *International Journal of Computer Vision*, 2, 251-282, 1989.