A Practical Constant Time Sorting Network

R. Lin[1]
Department of Computer Science
SUNY at Geneseo
Geneseo, NY 14454

S. Olariu[2]
Department of Computer Science
Old Dominion University
Norfolk, VA 23529

Abstract

We propose a novel VLSI sorting network implementing Leighton's column sort. Our network is mesh-based and modular; it consists of comparison-exchange processing elements (PEs), routing paths, and short broadcast buses. Each bus contains a small number of simple switches that we call shift switches. We enhance and simplify our previously proposed shift switching mechanism [9-11] to obtain an efficient O(1) VLSI-optimal sorting algorithm.

From a theoretical perspective, our new approach reduces significantly both the number of PEs (from N^2 to $N^{13/9}$) and the number of broadcasts from more than 58 bus broadcasts, each over N switches, to at most 16 bus broadcasts, each over $N^{4/9}$ switches.

From a practical standpoint, our network features a significant time-performance gain in comparison with the bitonic sorting circuit, especially when multiple smaller size arrays are sorted in parallel.

1. Introduction

Processor arrays with buses have become the focus of much interest due to recent advances in VLSI and fiber optic technology. Architectures featuring a reconfigurable bus system (REBS) including the bus automaton [16], the reconfigurable mesh [14], and the polymorphic-torus [7] allow the configuration of the corresponding bus system to be changed dynamically under program control, to suit communication needs. These architectures have been extensively investigated and many efficient algorithms have been proposed. Examples include several fundamental algorithms on sorting, tree search, image processing, computational geometry, vision, and graph theory [1, 4, 7-8, 12-17, 20].

Recently the authors have proposed a new reconfigurable bus system model [9-11]. Our idea involves enhancing traditional buses by the addition of a new feature that we call shift switching. The novelty of our idea is that we adopt a new class of switch states in a bus system. Specifically, we enable switches to rotate connections between lines of a bus. It turns out that this is a simple and powerful approach to improve the performance of a bus system.

The reconfigurable bus models were slow to gain wide acceptance because of its basic assumption, namely that the time needed to transmit a signal along a bus is a constant,

1 The work was supported, in part, by National Science Foundation under grant MIP-9307664
2 The work was supported, in part, by National Science Foundation under grant CCR-8909996

0-8186-3492-8/93 $3.00 © 1993 IEEE

regardless of the number of switches through which the signal propagates. According to traditional semiconductor technology, it is true that the transmission rate of a switch has a lower bound, however, recent VLSI implementations have demonstrated that the broadcast delay incurred by traversing one switch is indeed quite small. For example, a switch delay obtained on a YUPPIE chip is about (1/16)ns to 1ns (see [13] pp. 238). This delay is even shorter on a recent chip called GCN, which adopts pre-charged circuits [17]. This confirms the feasibility and potential benefits of the models, particularly, when there is no (or only few) long distance broadcast required in the computation.

The purpose of this paper is to propose a VLSI sorting network involving only broadcasting over "short" buses. We view our contributions at two levels. On the theoretical side, our approach significantly reduces the number of broadcasts involved in the previously known O(1) sorting algorithms [4, 12, 15] on a 2-D REBS. Also, and perhaps more significantly, our architecture is practical to implement, thus competing with the best known existing sorting networks, such as, bit serial bitonic sorting circuit (B-CIRCUIT for short) [2]. Sorting N data items in O(1) time on a REBS has been previously studied by several researchers. Among them, Wang et al. [20] present a simple algorithm on a reconfigurable $N\times N\times N$ mesh. Ben-Asher et al [1] propose an O(1) sorting algorithm on a reconfigurable cube (mesh of meshes) which can be embedded on a chip of size $X\times Y\times Z=N\times N\times(3+4k)$ (the depth of construction recursion $k\geq 2$), using $N^{1+\epsilon}$ processors for $1>\epsilon>0$. The algorithm implements Leighton's column sort, however, it contains a large constant time factor and is far from practical. Recently Lin *et al.* [12] and V.K. Prasanna and J. Jang [4] present two results, based respectively on selection sort and Leighton's column sort to reduce the number of processors required for the sort to $N\times N$. The algorithms in [4, 12] have achieved AT^2 lower bound in VLSI word model. However, both these algorithms are rather involved, requiring more than 100 broadcasts on buses of N switches. Quite recently, M. Nigram and Sahni have presented an improved version of the constant time sorting algorithm, which further reduces the number of bus broadcasts to only 59 [15].

The sorting algorithms proposed in this paper are also based on Leighton's column sort. However, our sorting networks utilize the shift switching mechanism, achieving a far more efficient O(1) sort. Our network is a 1- or 2- level embedded mesh structure (it can also be extended to k-levels). Our 1-level sorting mesh improves on the previous results in two aspects: first, it requires only $N^{5/3}$ PEs[3]; second, it requires only 8 bus broadcasts on buses with a *cycle (the number of switches a bus contains)* of $N^{2/3}$, 6 broadcasts on paths of no switches; 2 broadcasts involving $N^{1/3}$ switches, and 6 comparisons. It is important to note that for broadcasting purposes we distinguish between buses and paths, specifically, a bus contains switches, whereas a path dose not.

A particularly attractive property of our sorting network is the way in which it handles the task of sorting multiple arrays in parallel. With a larger fixed-size sorting network we can organize sorting multiple smaller arrays in parallel very efficiently. For example, with our 2-level sorting network of maximum input size $N=2^{13}$, we can sort and output 2^7 arrays in parallel, each having 2^6 item, in 1 comparison, 1 bus broadcast (over 64 switches) and 2 (routing) path broadcasts (with 64 PEs along the path listening); also we can sort 2^5 arrays in parallel, each having 2^8 items, in 6 comparisons, 4 bus broadcasts, and 8 path broadcasts; finally, we sort 2^{13} items, in 26 comparison, 16 bus broadcasts, and 30 path broadcasts.

Compared with B-CIRCUIT, our networks feature a significant time gain, particularly, when multiple smaller arrays are sorted in parallel. To sort an array of N (for $2^9\leq N\leq 2^{18}$)

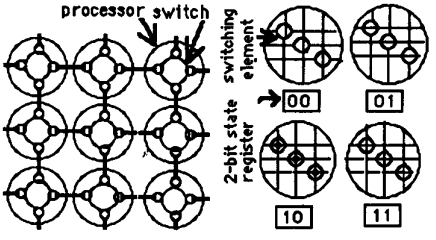
[3] In this paper by j^x (or $\log_m N$) we mean the smallest integer not smaller than j^x ($\log_m N$)

items of logN bits, we exhibit a time gain of 1.6 to 1.9. To sort multiple (16 to 1024) smaller input arrays of total size of N, we have a time gain from 6.5 to 10.5. Our network requires a larger area than the corresponding B-CIRCUIT only by a factor of O(logN).

The remainder of the paper is organized as follows: Section 2 reviews the concept of shift switching on a reconfigurable bus system. Section 3 presents two shift switching bus architectures for our processor assignment problem. Section 4 presents the 1-level sorting network and the algorithm. Section 5 introduces the 2-level sorting network and the algorithms. Section 6 compares our sorting networks with B-CIRCUITs. The final section concludes the paper.

2. Shift switches for reconfigurable bus systems

In this section we briefly review the basic features of a reconfigurable mesh and some concepts on shift switching which have been introduced in [9-11]. A reconfigurable mesh consists of an N×N VLSI array of PEs overlaid with a reconfigurable bus system (Figure 1a). Every PE features a switch with four ports denoted by N, S, E and W. Local connections between these ports can be established by setting up switch states under program control. The regular structure of the reconfigurable mesh makes it suitable for VLSI implementation. For our purposes, a switch (Figure 1.b) of REBS can be seen as an array of m identical switching elements, which are under synchronous control of a PE. A shift switch can be constructed from simple switches primarily through reorganizing its internal wiring (see Figure 2). It ensures that shift (or rotation) connections between the incoming and outgoing bus lines can also be dynamically established. The notation $S_{m:d}$ stands for a shift switch featuring m switching elements, with the (switch) state changes controlled by d bits. Equipped with an $S_{m:d}$ switch a processor can shift one (or zero) bit of an incoming m-bit signal. We assume the following: (1) Each switch has a d-bit register called state register: if the contents of this register is k, then the processor can set its switch to state k; (2) A switch has an AND gate with two inputs a, b and an output c (called *rotation bit*). When a processor sets its switch to state: 0, the contacts w and e in each switching element are connected; 1, the contacts w and i in each switching element are connected; It is easy to confirm that in state 1 (resp. 0) the incoming signal is shifted 1 (resp. 0) bit. One of our basic results is given below: *Summing N -1 bits (or N bit with at least one of them is 0) can be computed in* $(\log_m N)\delta(N)$ *time using a linear array of N shift switches of* $S_{m:1}$ *plus an m-to-logm encoder and a shift register* (refer to Theorem 1 of [9]).



(a) A 3×3 REBS mesh (b) A switch and 4 switch states

Figure 1. Reconfigurable bus system.

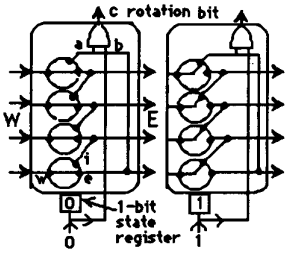
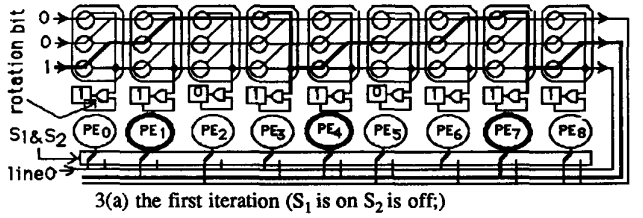


Figure 2. Two-port shift switch $S_{4:1}$.

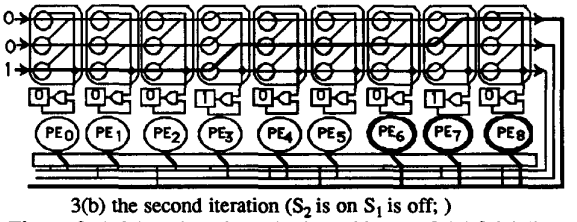
3. The processor assignment problem and broadcast buses

For the purpose of this paper, we describe two shift switching broadcast bus architectures to solve the following problem termed *the processor assignment problem*: Consider an array of k processing elements (PEs): PE_0, .. ,PE_{K-1}. The input is a bit sequence b_0, .. ,b_{K-1} stored one bit per PE, with PE_i storing b_i for all i. Let SUM=b_0+ .. +b_{K-1}≤K-1, the task is to assign PE_{SUM} with a bit 1, PE_j (j≠SUM) with a bit 0 (note our applications ensure SUM≤K-1).

The first proposed architecture is called *2-broadcast bus* (see Figure 3). It consists of a bus of K shift switches $S_{m:1}$, for $m=K^{1/2}$, PE_i is associated with i-th $S_{m:1}$. There is a line to connect the rotation bit of each switch to its state register. The bus has m extension lines (indexed from 0 to m-1, called "feed-back" lines) connecting to PEs by two switch arrays (S1 and S2) in the following way: through S1 each P_i connects to the (i MOD m)-th line, through S2 each PE_i connects to the (i DIV m)-th line. Initially, we load the state register of each switch with the corresponding input bit. Then, the following sequence of steps is performed twice (called *assignment algorithm-1*): (1) Set up shift switches according to the values in their state registers, and turn on S1 (S2) turn off S2 (S1) in the first (second) iteration; (2) Broadcast an m-bit signal 10..0 (called *shifting signal*) from the west port of the bus; (3) Each PE receives a bit from the feed-back line, and each state register is loaded with a new value through the associated AND gate (Figure 3). Now, for each PE the AND of the two bits received is the result bit (or the assigned bit). The correctness of the algorithm can be seen by noticing that the unique PE_i received a bit 1 twice satisfies: i MOD m=SUM m and i DIV m=SUM DIV m (i.e. i=SUM). With a careful design, the running time[4] of the algorithm is about $2TB_k$. The number of switch elements required is $k^{1/2}$+1 per PE.



3(a) the first iteration (S_1 is on S_2 is off;)



3(b) the second iteration (S_2 is on S_1 is off;)

Figure 3. A 2-broadcast bus with input bits (1,1,0,1,1,0,1,1,1).

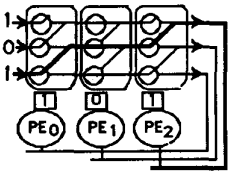


Figure 4. 1-broadcast bus.

The second architecture is called *1-broadcast bus* (see Figure 4). It consists of a bus of K shift switches $S_{k:1}$, PE_i is associated with i-th $S_{k:1}$. The bus has k "feed-back" lines, connecting to PEs, such that, PE_i connects to the i-th feed-back line. The *assignment algorithm-2* is very simple: After an initial process similar to that for the 2-broadcast

[4] We use TB_x to denote the delay time of a signal propagation through x switch elements

architecture, we broadcast a k-bit shifting signal; The feed-back bit received by a PE is the result bit. The correctness of the algorithm is obvious. The running time is TB_k. The number of switch elements used is k per PE. The algorithm runs faster but use more switches.

4. 1-level sorting network

In this section we present the details of our 1-level sorting network. The construction involves two steps. In Step 1, we construct an initial mesh with N×N *row paths* and *column paths* of logN bits, connected only at the main diagonal nodes (see Figure 5). In Step 2, we partition the mesh into N^a submeshes of $N \times N^b$. We assume that both N^a, N^b are integers and that a+b =1, b ≥2a>0. An on-off inter-submesh switch is added on each path between two successive submeshes, and all these switches have identical on/off state at any moment. For every submesh, we divide the row paths into N^b groups, N^a paths per group. For each group, we add a 2-broadcast bus of N^b PEs, one PE being associated with a column (Figure 6). We refer to the PEs residing on the main diagonal of each submesh as major PEs. For each bus we add two special row path segments of width logN bits each, called T-path segment, and B-path segment, one above and one below the bus. Each PE residing at the right end of a bus connects to its right neighbor PE (if such a neighbor exists).

Clearly, all major PEs form an $N^b \times N^a$ matrix. We call this matrix as data matrix. We now assign two types of global indices to the data matrix as follows: a column-major index (shown in bold) and a row-major index (shown in plain). For each submesh we also index all PEs with (local) matrix indices (i, j), 0<=i, $j<=N^{b-1}$. Note that since there exists 1 to 1 correspondence between major PEs and buses or T- or B- path segments, they form the same type of matrices of their own, so the indices described above will also refer to them. Submeshes are marked as even and odd successively (each PE knows the value). We connect the T-path segment with row-major-index i to row path i (as shown by vertical lines in bold), and connect each B-path segment to the associated column path (i.e. the column path that corresponds to the same major PE as the B-path does).

The 1-level sorting mesh has total N^{1+b} PEs. Each PE (Figure 7) can read/write its three ports: T port (connecting to the T-path segment), B port (connecting to the B-path segment), and C port (connecting to the associated). The PEs residing at the right ends of buses can read/write data from their right neighbors, through the connection path, if such neighbors exist. Each PE has a logN-bit comparator (see Figure 7, 8) which is a sequentially connected array of logN 1-bit comparators. In fact, the structure can be seen as of logN exclusive OR gates and 2logN on-off switches (it is on if control bit is 1, off otherwise). The comparator receives two logN-bit unsigned numbers a and b, outputs 0 if a<b, 1 if b<a, x if a=b, here x is determined by the PE's local index (i, j) as: 1 if i<j, 0 otherwise. The purpose of using such a comparator is that it also utilizes "short" broadcasting. the comparator delay is about $T_c + TB_{logN}$, here T_c is the 1-bit comparator's delay time. (Note it is easy to obtain a slightly modified such a comparator for the input of 2s complement numbers).

The following important facts should be noticed: (a) When the major PE with row major index i (Figure 9.a) writes a message to its T-path-segment through its T port, if the inter-submesh switches is on, the message will propagate along the row path i to column path i and be received by the PE with column major index i through its C port. Thus it takes one *path-broadcast* to transpose the data matrix. In fact, the message can also be received by all PEs in the same column of the submesh. (b) Similarly, the reverse permutation can also be done in 1 path-broadcast (and the broadcast message can also be received by all PEs of the whole bus). (c) When inter-submesh switches are off, major PE(i, i) can distribute a data item to

PEs with local row index i and to PEs with local column index i by writing it to C port and then letting all those PE read it from B-port and Cport respectively (Figure 9.b).

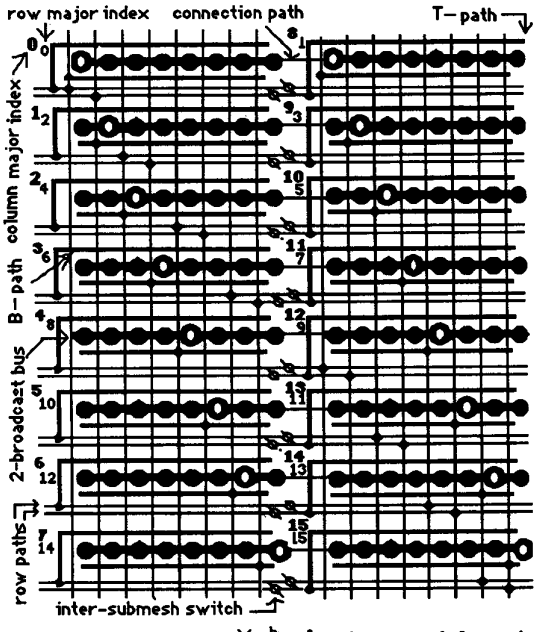


Figure 6. The 1-level $(N \times N^b)$-N^a sorting network for sorting $N=2^4$ items, here b=3/4, a=1/4. ●: major PE, ●: non-major PE.

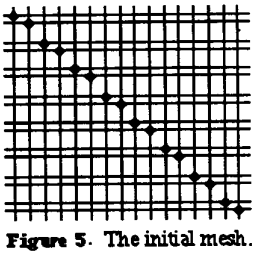


Figure 5. The initial mesh.

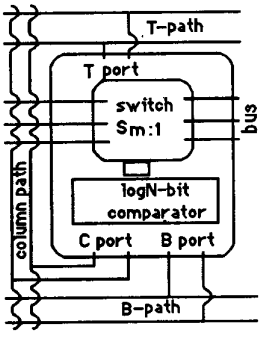


Figure 7. PE structure

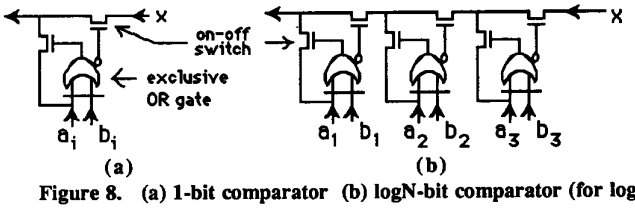


Figure 8. (a) 1-bit comparator (b) logN-bit comparator (for logN=3).

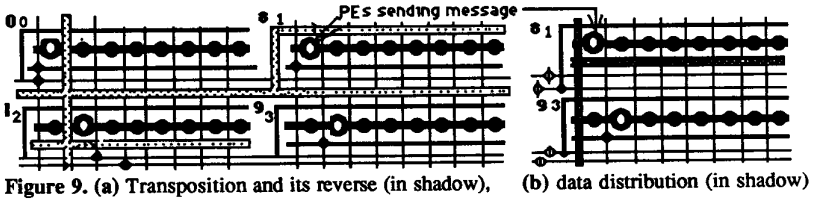


Figure 9. (a) Transposition and its reverse (in shadow), (b) data distribution (in shadow)

We use the 1-level sorting network to implement the seven phases of Leighton's column sort [6], which can be used to sort a r×s matrix where $r \geq s^2$. N (= rs) elements are sorted in column major order after the following 7 phases are executed: during Phases 1, 3, 5, and 7, the items are sorted within each column smallest-first (except in Phase 5, where adjacent columns are sorted in reverse order). During Phases 2 we "transpose" the matrix by picking up the items in column-major order and setting them down in row-major order (preserving the r×s shape). During Phase 4 we reverse the permutation applied in Phase 2, picking up the items in row-major order and setting them down in column-major order. During Phase 6, we apply two steps of odd-even transposition sort to each row.

Our implementation closely follows these steps with as many operations merged in one instruction as possible. In Phases 1, 3, 5 and 7, sorting columns of the data matrix is implemented by rank sort on each submesh, which involves the following operations: distributing the data matrix to corresponding rows and columns of the submesh (see Figure 9.b); comparison; assigning PEs with 1 and 0 bits (see section 2); the unique PE assigned bit 1 send the ranked item to the column, thus the data on the submesh columns are sorted. In Phase 2 and 4, the inter-submesh data movements are obtained by utilizing the directly connected path structure (see Figure 9.a). The following is the algorithm on our 1-level sorting network (see Figure 6):

Algorithm SORT1 (initially the inter-submesh switches are off)
Input: N items on the column paths (they are automatically distributed and data matrix is formed). Output: N sorted items in smallest-first on the column paths.
For each PE with a local index (i, j) PARDO:
Phase 1. {rank sort the data matrix columns}
1.1. Compare the item (referred to as X) in B port and item (referred to as Y) in C port.
1.2. Write the state register 1, if (X>Y) or (X=Y and j>i), 0 , otherwise.
1.3. Apply the assignment algorithm-1.
1.4. (turn on the inter-submesh switches) The allocated PE sends the item in B port to C port. {the items in the column paths are the sorted outputs of submeshes}
Phase 2. {data matrix transposition: It is already done and data are in T-ports}
Phase 3. {rank sort the data matrix columns} (turn off the inter-submesh switches)
3.1. If i=j, send the item of T port to C port. 3.2. Repeat 1.1 to 1.3.
3.3. The allocated PE send the item in B port to C port.
Phase 4. {permute the data matrix reversely} (turn on the inter-submesh switches)
4.1. If i=j, send the item of C port to T port.
Phase 5. {rank sort the data matrix columns with adjacent columns sorted in reverse order}
5.1. (turn off the inter-submesh switches) Compare the data item X of B port and the item (referred to as Y) in C port.
5.2. If j is even, writes the state register 1, if (X>Y) or (X=Y and j>i), 0, otherwise.
If j is odd, write the state register 0, if (X>Y) or (X=Y and j>i), 1, otherwise.
5.3. Repeat 1.3, 1.4 (the item in B port is now referred to as X).
Phase 6. {apply two steps of odd-even sort to each row of the data matrix}
6.1. If the PE's submesh is marked even, and the PE is in the last column of the submesh, compare X and its right neighbor's X (if there is one), and write the smaller back to its right neighbor PE, then update its X with the larger; the right neighbor PE updates its X.
6.2. Repeat 6.1. for the submeshes marked odd.
Phase 7. {rank sort the data matrix columns}
7.1 If the PE is the first PE send X to B port.
7.2. Repeat 1.1 to 1.4, the data items in the column paths are the sorted output.

Clearly, the algorithm calls assignment algorithm-1 for 4 times, requires 6 comparisons

on logN-bit items, uses 6 times path broadcasts for distribution of data, It also requires 2 times "long" path broadcasts, each involving N^a inter submesh switches, for transposition and reverse permutation of data. In this paper we measure the distance of a broadcast by the number, j, of basic hardware elements (defined as comparator bits, or switching elements) plus the number, k, of wire cross sections, whose total physical length are equal to the distance, denoted by $E_j + W_k$. We use TE_j and TW_k to represent the time for broadcast on paths of distances of E_j, and W_k respectively. In summary, the sorting time is:

$T1=8TB_{N^b}+6(T_c+TB_{\log N})+6(TE_{N^{3b/3}}+TW_{N\log N})+2(TE_{N^{3b/2}}+TE_{N\log N}+TW_{2N\log N}+TB_{N^a})$ ..(A)

Theoretically, T1 is a constant (since each broadcast is assumed as $O(1)$. The required VLSI area (word model) is $O((N+N^b \cdot N^{b/2}/\log N)*N=O(N^2)$ for b=2/3. Thus the 1-level sorting network is AT^2 optimal. Also it requires only $N^{2b}*N^a=N^{1+b}$ ($=N^{5/3}$, if b=2/3) PEs, and $(N^b)^{1/2}+1$ ($=N^{1/3}+1$, if b=2/3) switching elements per PE.

5. 2-level sorting network

In this section we present the details of our 2-level sorting network by modifying the 1-level sorting network (see section 4) to a 2-level network. It involves two steps. In step 1, we replace each 2-broadcast bus by a special PE called sub-PE and locate it at the original major PE position. Each T-path segment is shortened and re-positioned to be dedicated to the sub-PE, and each B-path segment is renamed to be the row path of the second level. (Figure 10.a). Each sub-PE takes over two roles of the removed major PE: (1) read/write the corresponding column and T-path for transposition and reverse permutation; (2) read/write its right neighbor sub-PE through the connection path and compare two data items (for the two steps of odd-even sort). It also can read/write the corresponding new row path, but it does not involve summing bits, so has no shift switch associated with it (see Figure 10.b). Now the second level row paths that line up vertically and the corresponding column paths form a second level initial sorting mesh (total N^a such initial meshes) (see Figure 10.c).

In step 2, each second level initial mesh is further constructed in a way the same as that for Step 2 of 1-level network construction (Figure 10.d), except that (1) no inter-submesh switch is added; (2) 1-broadcast buses are used, in stead of, 2-broadcast buses; (3) on-off inter-submesh switches are not used in this level (because each second level submesh is seen as a basic module). Figure 11 shows the layout of the complete 2-level sorting network reconstructed from the 1-level network of Figure 5.

We present two sorting algorithms, Sort_2nd(r) (as a subroutine) and SORT2. Each algorithm is a close implementation of Leighton's seven-phase column sort. For Sort_2nd(r), if r =1 the sorted output is in smallest-first, if r=0 the output is in reverse order.

Algorithm Sort_2nd (r=1)

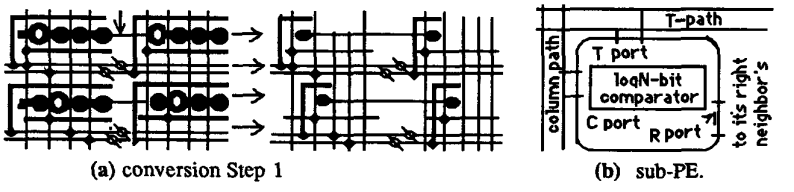
Input: N^a arrays, with N^b items each, distributed on the column paths.

Output: ; N^a smallest-first sorted arrays, with N^b items each, distributed on the column paths (the arrays relative position not changed after sorting) .

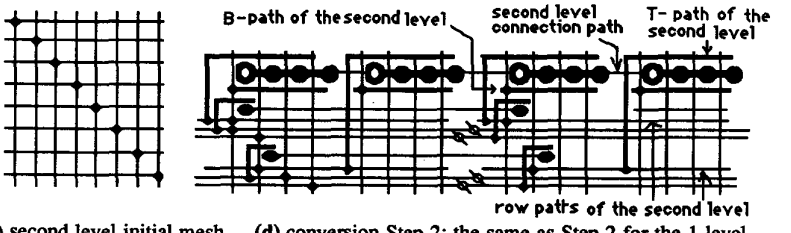
For each submesh apply algorithm SORT1 with two modifications: (1) Apply the assignment algorithm-2; (2) Remove all statements related to inter submesh switches). Algorithm **Sort_2nd(r=0)** can be obtained from this version by changing all statements of "writing x into the state register" to "writing the complement of bit x into the state register".

Algorithm SORT2 (initially the inter-submesh switches are off)

Input: N items on the column paths. Output: N sorted items on the column paths.



(a) conversion Step 1 (b) sub-PE.



(c) second level initial mesh (d) conversion Step 2: the same as Step 2 for the 1-level.

Figure 10. The conversion of 1-level sorting network of Figure 5 to 2-level sorting network.

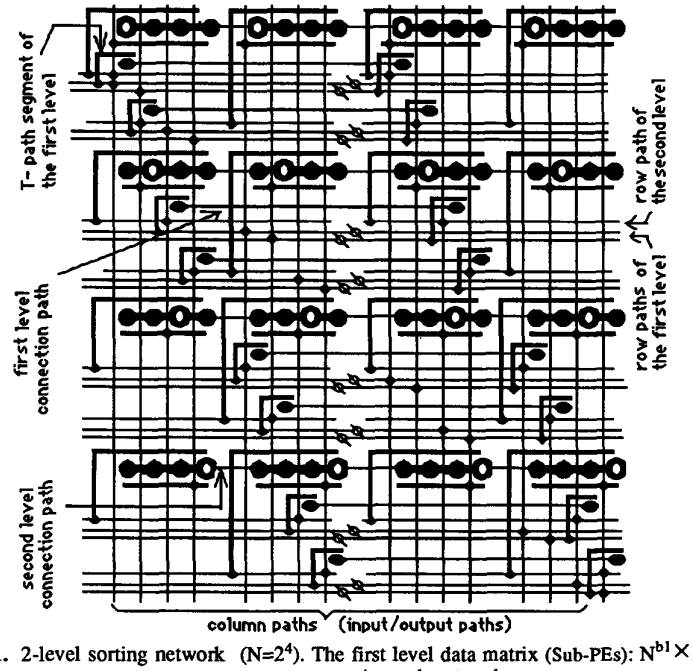


Figure 11. 2-level sorting network ($N=2^4$). The first level data matrix (Sub-PEs): $N^{b1} \times N^{a1} = 2^3 \times 2^1$, the second level data matrix (major PEs): $N^b \times N^{a1} = 2^2 \times 2^1$, ● representing sub-PE.

Phase 1. {sort each column of the data matrix}
1.1. Input data items from column paths (data are distributed and data matrix is formed).
1.2. Sort_2nd(1) {note we turn on the inter-submesh switches before the last path broadcast}
Phase 2. {data matrix transposition: It is already done and data are in T-ports of sub-PEs}
Phase 3. {sort each column}
3.1. (turn off the inter-submesh switches) For each sub-PE send the item in T port to C port.
3.2. Sort_2nd(1)
Phase 4. {permute the mesh reversely} (turn on the inter-submesh switches)
For each sub-PE send the item in C port to T port. (turn off the inter-submeshswitches).
Phase 5. {sort columns with adjacent columns sorted in reverse order}
5.1. Apply Sort_2nd(1) (Sort_2nd(0)) on 2nd-level meshes marked even (odd).
Phase 6. {apply two steps of odd-even sort to each row}
6.1. For 2nd-level meshes marked even, each sub-PE compare item (X) in C port and the item in R port (the right neighbor sub-PE's X, if there is such a neighbor), and write the smaller back to R port (to its right neighbor's C port), then update its X value with the larger; the right neighbor sub-PE updates its X with the received value.
6.2. Repeat 6.1 for 2nd-level meshes marked odd, and send updated X to C port.
Phase 7. {sort each column} 7. Sort_2nd(1).

In summary, algorithm SORT2 calls assignment algorithm-2 for 16 times. It requires 26 comparisons on logN-bit items, uses 4*7=28 times path broadcasts for distribution of data, It also requires 2 times "long" broadcasts, each involving N^a inter submesh switches, for transposition and reverse permutation of data. Thus, the sorting time can be described as:

$$T2=16TB_{N^b}+26(T_c+TB_{\log N})+28(TE_{N^{2b}}+TW_{N\log N})+2(TE_{N^{2b}}+TE_{N\log N}+TW_{2N\log N}+TB_{N^a}) \quad ..(B)$$

Similar to 1-level sorting network, theoretically, our 2-level network requires a constant for the sorting. The VLSI area (word model) is $O((N+N^b \cdot N^b/\log N)\cdot N=O(N^2)$ for b=4/9. Thus, the network is AT^2 optimal. Also the sorting network requires only N^{1+b} ($=N^{1+4/9}$, if b=4/9) PEs, and N^b($=N^{4/9}$, if b=4/9) switching elements per PE.

6. Comparison with bitonic sorting circuits

One important feature of our sorting networks is the way in which our network handles the task of sorting multiple arrays in parallel. With a larger fixed-size sorting network we can organize sorting multiple smaller arrays in parallel very efficiently, because the sorted subarrays are also resided in the column paths and the arrays' relative positions are unchanged during the sorting. For example, with a 2-level sorting network of maximum input size $N=2^{13}$, we can sort and output 2^7 arrays in parallel, each having 2^6 item, in 1 comparison, 1 bus broadcast (over 64 switches) and 2 (routing) path broadcasts. We denote the sorting time by T2 (2^7, 2^6). Also we can sort 2^5 arrays in parallel, each with 2^8 items in T2(2^5, 2^8) which consists of 6 comparisons, 4 bus broadcasts, and 8 path broadcasts. Finally, we sort 2^{13} items in T2(1, 2^{13}) whth 26 comparisons, 16 bus broadcasts, and 32 path broadcasts.

We compare the sorting networks with B-CIRCUIT of depth logN(logN+1)/2 [2]. Because the longest path broadcast delay is TW_N (vertical direction) + TE_N (horizontal direction), B-CIRCUIT requires sorting time: $T0=\log N(\log N+3)/2\ (T_c+TW_N+TE_N)$ ..(C)

We examine three input sizes $N=2^9, 2^{13}, 2^{18}$ for the network time gain, G, over B-CIRCUIT.

$N=2^9$: For the input of this size or smaller, we use 1-level network, a mesh of $N=2^6 x 2^3$ (b=2/3, a=1/3, $N^b=2^6$, $N^a=2^3$). By algorithm SORT1 and (A) we have,

$$T1(2^3, 2^6)=2TB_{64}+(T_c+TB_9)+2(TE_{N^b}\cdot N^{b/2}+TW_{N\log N}), \text{ and}$$

$T1(1, 2^9)=8TB_{N^b} + 6(T_c+TB_{logN}) + 6(TE_{N^b}.N^{b/2} + TW_{NlogN}) + 2(TE_{N^b}.N^{b/2} + TE_{NlogN} + TW_{2NlogN}+TB_{N^a})$. Let $TX=(T_c+TE_N+TW_N)$, we have $T0(2^3, 2^6)=27TX$ and $T0(1,2^9)=54TX$. (assuming that B-CIRCUIT has multiple output ports).

Based on today's VLSI technique and the results of analyses [3, 7, 8, 13, 14, 17], in this paper we assume the following: (1) One bit comparison takes 1 time unit (say 5ns to 100ns), i.e. $T_c=1$; (2) Propagation delay on a short broadcast bus of cycle 64 switches takes 2 time units, i.e. $TB_{64}=2$, and $TB_9=1$; (3) Propagation delay on a path (containing no swich) of moderate length takes 1.5 time unit, i.e. $TE_N=TE_{512}=1.5$ and; (4) Propagation delay (including delay caused by VLSI packaging) on a path or on a bus (when its cycle is greater than 64) is linear on distance (i.e. $TE_{NlogN}=logNTE_N$, $TB_{256}=4TB_{64}$. We also assume that the width of a basic element is about logN (or 15 in general) times as larger as that of a wire, i.e. $TW_N=TE_N/15=0.1$. Thus $T1(2^3, 2^6)=10.8$, $T0(2^3, 2^6)=70$, G=6.5

$T1(1, 2^9)=75$, $T0(1, 2^9)=140$, G=2

$N=2^{13}$: For the input of this size or larger, we use 2-level network. Our network now is a mesh of $N=(2^6x2^3)x2^4$ (b=6/13, a1=3/13,b1=9/13, a=4/13, $N^b=2^6$), by algorithm SORT2 and (B) we have, $T2(2^7, 2^6)= TB_{N^b}+ (T_c + TB_{logN})+ 2(TE_{N^{2b}}+TW_{NlogN})$,

$T2(2^4, 2^9)=4TB_{N^b} + 6(T_c + TB_{logN})+ 6(TE_{N^{2b}}+TW_{NlogN})$, and

$T2(1, 2^{13})=16TB_{N^b}+26(T_c+TB_{logN})+28(TE_{N^b}.N^b+TW_{NlogN})+2(TE_{N^b}.N^b+TE_{NlogN}+TW_{2NlogN}+TB_{N^a})$. We also have, $T0(2^7, 2^6)=27TX$, $T0(2^4, 2^9)=54TX$, and $T0(1, 2^{13})= 104TX$. thus, $TB_{N^b}=TB_{64}=1$, $TE_{N^{2b}}=TE_{N^{12/13}}=2^3*TE_{2^9}=8$, $TE_N=TE_{2^{13}}=16$, $TE_{NlogN}=13*16=208$, $TW_N=16*0.1=1.6$, $TW_{NlogN}=13*1.6=20.8$, and $TX=(T_c+TE_N+TW_N)=18.6$.

$T2(2^7, 2^6)= 69$, $T0(2^7, 2^6)=718$, G=10.5

$T2(2^4, 2^9)=216$, $T0(2^4, 2^9)=1436$, G=6.6

$T2(1, 2^{13})=1735$, $T0(1, 2^{13})=2766$, G=1.6

$N=2^{18}$: Our network now is a mesh of $N=(2^8x2^4)x2^6$ (b=4/9 $N^b=2^8$). Similarly, we have $TB_{N^b}=TB_{256}=4TB_{64}=8$, $TE_N=2^9TE_{2^9}=512$, $TE_N=128$, $TW_N=2^9TW_{2^9}=51.2$, TX=564.

$T2(2^{10}, 2^8)=4464$, $T0(2^{10}, 2^8)= 36088$, G=8.1

$T2(2^6, 2^{12})=12239$, $T0(2^6, 2^{12})=73818$, G=6.1

$T2(1, 2^{18})=88790$, $T0(2^7, 2^6)=155017$, G=1.7

The above analyses reveal that using our sorting networks to sort an array of N (for $2^9 \le N \le 2^{18}$) items of logN bits, we have a time performance gain of 1.6 to 2, to sort multiple (16 to 1024) smaller input arrays of total size of N, we have a time gain about 6 to 10. Our network has a length of $N*(E_{logN}+W_{logN})$ and high of NE_1+NW_{logN} (1-level network) or $N^{8/9}E_1+NW_{logN}$ (2-level network). For either case the area is about $O(N^2*E_1E_{logN})$. The B-CIRCUIT has an area $O((E_1log^2N+ qNW_1)*NE_1)$, here q=2 (or 3). In general, our network requires a larger area than B-CIRCUIT by a factor of 3logN to 5logN.

7. Conclusions

The main contribution of this work is to propose a practical VLSI sorting network with limited broadcasts and reconfiguration. Our approach not only significantly improves the previously known O(1) sorting algorithms theoretically [1, 4, 12, 15], it also shows attractive features making our O(1) sorting networks competitive with the best known practical sorting networks, such as bit serial bitonic sorting circuit with depth logN(logN+1)/2 (B-CIRCUIT).

Sorting N data items in O(1) time on a reconfigurable mesh has been studied by several

researchers [1, 4, 12, 15]. However, all known algorithms for the problem are rather involved, requiring more than 58 broadcasts on buses of N switches. The new sorting network uses Leighton's column sort, and adopts and simplifies our previously proposed shift switching mechanism [9-11], thus achieving a much more efficient O(1) VLSI AT^2 optimal sorting.

Our network is mesh based. One important feature of our sorting networks is that it can easily achieve a very small (say, $N^{2/3}$ to $N^{4/9}$, or 64 to 256 for N from 2^9 to 2^{18}) bus cycle (the number of switches contained by the bus). Because of small bus cycle each bus can be implemented practically in O(1) time. In addition to smaller bus, our sorting network is also modular. Specifically, all data movement operations are local to a small module, except for two path broadcasts requiring a distance of the network diameter.

A particularly attractive property of our sorting network is the way in which it handles the task of sorting multiple arrays in parallel. With a larger fixed-size sorting network we can organize sorting multiple smaller arrays in parallel very efficiently. Compared with the bitonic sorting circuit, our networks feature a significant time-performance gain, specifically, when multiple smaller size arrays are sorted in parallel. Based on the analysis under a reasonable assumption (propagation delay, for "short" broadcast is a constant, in general, is linear on the broadcast distance), we find that to sort an array of N (for $2^9 \le N \le 2^{18}$) items of logN bits, we have a time performance gain about 1.6 to 2, to sort multiple (16 to 1024) smaller input arrays of total size of N, we have a time gain within a range of 6 to 10. Also our network requires a larger area than B-CIRCUIT only by a factor of 3logN to 5logN.

References

[1] Y. Ben-Asher, D. Peleg, R. Ramaswam, and A. Schuster, The power of reconfiguration, *J. of Parallel and Distributed Computing*,vol.13, 1991, 139-151.

[2] K. E. Batcher, Sorting networks and their applications, *Proc. AFIPS Conf.*, 1968.

[3] G. Bilardi, M. Pracchi, and F. P. Preparata, A Critique of network speed in VLSI models of computation, *IEEE Journal of Solid-State Circuit*, Vol. SC-17, NO. 4, august 1982.

[4] J. Jang and V. K. Prasanna, An optimal sorting algorithm on reconfigurable mesh, *Proc. of International Parallel Processing Symposium (IPPS)*, March, 1992.

[5] F. T. Leighton, Tight bounds on the complexity of parallel sorting, *IEEE, Trans. Compt.*, C-34, 4, 1985.

[6] F. T. Leighton, *Parallel algorithms and architectures: Arrays.trees.hypercubes*, (Morgan kaufmann publishers, 1992), 261.

[7] H. Li and M Maresca, Polymorphic-torus network, *IEEE Trans. Comput.*,38 (9) (1989) 1345-1351.

[8] H. Li and Q. Stout, *Reconfigurable massively parallel computers*, Prentice-Hall , 1991.

[9] R. Lin, Reconfigurable Buses with Shift Switching -- VLSI Radix sort, *Proc. of Int.Conference on Parallel Processing* (ICPP), The Pennsylvania State Univ, Press, Chicago, Ill. 1992, Vol III, 2-9.

[10] R. Lin, S. Olariu, Computing the inner product on reconfigurable buses with shift switching, *Proc. of Joint Conf. on Vector and Parallel Processing*, (CONPAR 92), France, Sep 1992, 181-192.

[11] R. Lin, S. Olariu, Reconfigurable buses with shift switching architectures and applications, *Proc. of International Phoenix Conference on Computers and Communications* (IPCCC93), Mar, 1993 23-29 .

[12] R. Lin, S. Olariu, J. Schwing, and J. Zhang, Sorting in O(1) time on an n*n reconfigurable mesh, *Proc.of Ninth European Workshop on Parallel Computing* (EWPC),Spain, Mar, 1992. 16-27.

[13] M. Maresca and H. Li, Connection autonomy and SIMD computers: a VLSI implementation *J. of Parallel and Dis. Computing*,vol. 7, 1989 302-320.

[14] R. Miller, V. k. Prasanna Kumar, D. Reisis and Q.F. Stout, Mesh with reconfigurable buses, *Proc. 5th MIT Conference on Advanced Research in VLSI* (1988) 163-178.

[15] M. Nigram and Sahni, Sorting n numbers on NxN reconfigurable meshes with buses, *Proc. of Int. Parallel Processing Symp.*, Apr. 1993.

[16] J. Rothstein, Bus automata, brains, and mental models, *IEEE Trans. on SMC* 18, 1988.

[17] D. B. Shu and J. G. Nash, The gated interconnection network for dynamic programming, S. K. Tewsburg et al. (ed), *Concurrent Computations*, Plenum Publishing Corp., 1988.

[18] H. S. Stone, Parallel processing with the perfect shuffle, *IEEE Trans. on Compt.*vol. C-20, 2, 1971.

[19] C. Thompson, A complexity theory for VLSI, PhD dissertation CMU-CS-80-140, Comput. Sci. Dept. Carnegie-Mellon Univ. Pittsburgh, PA, Aug. 1980.

[20] B. F. Wang, G.H. Chen, and F.C. Lin, Constant Time Sorting on a processing array with a reconfigurable bus system, *Information Processing Letters*,vol. 34, no. 4, 187-192, 1990.