Optimal Sorting Algorithms for a Simplified 2D Array with Reconfigurable Pipelined Bus System

Min He, Xiaolong Wu, *Member*, *IEEE*, Si Qing Zheng, *Senior Member*, *IEEE*, and Burkhard Englert, *Member*, *IEEE*

Abstract—In recent years, many researchers have investigated optical interconnections as parallel computing. Optical interconnections are attractive due to their high bandwidth and concurrent access to the bus in a pipelined fashion. The Linear Array with Reconfigurable Pipelined Bus System (LARPBS) model is a powerful optical bus system that combines both the advantages of optical buses and reconfiguration. To increase the scalability of the LARPBS model, we propose a two-dimensional extension: a simplified two-dimensional Array with Reconfigurable Pipelined Bus System (2D ARPBS). While achieving better scalability, we show the effectiveness of this newly proposed model by designing two novel optimal sorting algorithms on this model. The first sorting algorithm is an extension of Leighton's seven-phase columnsort algorithm that eliminates the restriction of sorting only an $r \times s$ array, where $r \geq s^2$, and sorts an $n \times n$ array in $O(\log n)$ time. The second one is an optimal multiway mergesort algorithm that uses a novel processor efficient two-way mergesort algorithm and a novel multiway merge scheme to sort n^2 items in $O(\log n)$ time. Using an optimal sorting algorithm *Pipelined Mergesort* designed for the LARPBS model as a building block, we extend our research on parallel sorting on the LARPBS to a more scalable 2D ARPBS model and achieve optimality in both sorting algorithms.

Index Terms—Interconnection networks, optical networks, parallel algorithms and architectures, sorting.

1 Introduction

Recently, optical interconnection models have received extensive interest as building blocks for efficient parallel computing [1]. Optical signals transmitted on an optical waveguide have two important properties that are not shared with electrical signals running on an electrical bus, namely, unidirectional propagation and predictable propagation delays per unit length. These two properties enable high bandwidth and synchronized concurrent accesses of an optical bus in a pipelined fashion [2] and give optical interconnections great potential to improve the performance of algorithms designed for massive parallel processing systems. On the other hand, among different multiprocessor computing systems, reconfigurable multiprocessors [3], [4] using electronic interconnections have been investigated by many researchers for the following two reasons: first, reconfigurable systems are now well supported by Field Programmable Gate Arrays (FPGAs), which consist of up to millions of reconfigurable logic gates on a single chip; second, the possibility of changing the configurations of a system under program control allows using the same system to solve different problems. The above two facts have led to a tremendous research effort in designing systems that can exploit the advantages of both *optical interconnections* and *reconfigurations*. Linear arrays connected by pipelined optical buses, for example, are shown to be a powerful parallel computing model that provides the benefits of both. Two typical representatives of such systems are: the Linear Array with Reconfigurable Pipelined Bus System (LARPBS) [5] and the Array with Reconfigurable Optical Buses (AROBs) [6]. We are interested in designing parallel algorithms on a higher dimensional LARPBS: the 2D ARPBS. Many algorithms [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21] have been designed for the LARPBS model. Most of them outperform similar algorithms designed for multiprocessor systems using electronic interconnections, including the most widely studied one—the reconfigurable mesh [4].

Some previous research has been done to increase the power and robustness of the LARPBS model, including increasing the fault tolerance [12], [14] of the model, providing performance prediction of algorithms running on a realistic implementations of the model [22], and increasing the scalability of either the model or the algorithms running on the model [13], [19], [20], [21], [23]. The new model we propose in this paper aims at increasing the scalability of the LARPBS model. Because increasing the length of an optical bus introduces a lot of problems, such as longer bus cycles, power distribution, signal degradation, and synchronization error [24], scalability is an important issue for high-performance computing with optical interconnects. To increase the scalability of an optical interconnect, we can either scale algorithms designed for the model to run on a

- *M. He, X. Wu, and B. Englert are with the Department of Computer Engineering and Computer Science, California State University Long Beach, 1250 Bellflower Blvd., Long Beach, CA 90840-8302. E-mail: {mhe, xwu3, benglert}@csulb.edu.*
- *S.Q. Zheng is with the Department of Computer Science, Erik Jonsson School of Engineering and Computer Science, University of Texas at Dallas, 800 West Campbell Rd., EC 31, Richardson, TX 75080-3021. E-mail: sizheng@utdallas.edu.*

Manuscript received 26 Aug. 2008; revised 3 Mar. 2009; accepted 3 Apr. 2009; published online 17 Apr. 2009.
Recommended for acceptance by Y. Pan.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2008-08-0326.
Digital Object Identifier no. 10.1109/TPDS.2009.68.

smaller size model [19], [20], [21] or extend the model to higher dimensions [2], [25], [26], [27], [28], [29]. We focus on the second approach. Different two-dimensional optical bus systems, such as *array structure with synchronous optical switches* (ASOS) [2], *pipelined reconfigurable mesh* (PR-Mesh) [25], *array connected by pipelined optical buses with conditional delays* (ACPOB) [26], *symmetric ASOS* (SASOS) [27], *reconfigurable array with spanning optical buses* (RASOB) [28], and *array processors with pipelined buses using switches* (APPBS) [29], have been proposed in recent years. Among these architectures, ACPOB is the simplest and most powerful. An $n \times n$ 2D ACPOB has processors arranged as a grid with each row (respective column) connected by a pipelined TDM optical bus with conditional delays. Our simplified 2D ARPBS model is similar to a 2D ACPOB model except that we use an LARPBS for each row (respective column). In other words, the 2D ARPBS model has an extra set of switches, namely the reconfigurable switches, added for the convenience of reconfiguration. Although many parallel algorithms have been designed for the LARPBS model, few algorithms are designed for its two-dimensional extensions. That in turn motivates us to design optimal sorting algorithms for our newly proposed 2D ARPBS model.

Sorting is one of the fundamental problems in the field of computer science. In parallel computation setting, sorting can moreover be utilized for all kinds of rearrangements. Many different parallel sorting algorithms were developed for different models. For parallel sorting, mainly three parallel computation models have been considered—the circuit model, the PRAM abstract machine model, and the interconnection networks model. A parallel sorting algorithm that is based on comparison is said to be optimal (with respect to (shortly w.r.t.) the cost) if the number of comparisons is $O(n \log n)$ for sorting n items. The first $O(n \log n)$-time sorting algorithm is the AKS sorting network [30] on the circuit model and it is optimal as it uses $O(n \log n)$ comparators. Paterson simplified and improved the AKS network in [31]. But, he claimed in the paper that the constants obtained for the depth bound still prevent the construction being of practical value. The second optimal sorting algorithm is Cole's parallel mergesort [32] designed on the CREW PRAM model. Several parallel sorting algorithms [7], [8], [9], [10], [11] have been designed for the LARPBS model, including a constant-time parallel sorting algorithm that uses $O(n^2)$ processors [7], an efficient and scalable quicksort algorithm that runs in $O(\log n)$ average time and $O(n)$ worst-case time using n processors [8], an $O(\log^2 n)$-time deterministic algorithm that uses n processors [9]. In 2002, it was shown in [10] that sorting n items on an n-processor LARPBS can be done in $O(\log n \log\log n)$ time. Independently, we reported an optimal sorting algorithm of $O(\log n)$ time around the same time in [11].

1.1 Contributions

In this paper—to obtain better scalability while at the same time continuing to take advantage of optical interconnections and reconfigurations—we propose the 2D ARPBS model as an important extension of the LARPBS model. To prove the effectiveness of our new 2D ARPBS model, we furthermore develop two sorting algorithms for it and show their optimality. We use the optimal sorting algorithm *Pipelined Mergesort* [11] designed for the LARPBS model as a building block to construct these two optimal sorting algorithms. We present an optimal *Generalized Columnsort* and an optimal *multiway Mergesort* algorithm. Both sorting algorithms sort n^2 items on an $n \times n$ 2D ARPBS model in $O(\log n)$ time. The first sorting algorithm generalizes the idea of Leighton's seven-phase columnsort algorithm by introducing the concept of "*Virtual Matrix*" and "*Super Column*." While Leighton's algorithm only sorts an $r \times s$ array, where $r \geq s^2$, our new columnsort algorithm can sort an $n \times n$ regular shape 2D ARPBS. The second optimal sorting algorithm is a *Multiway Mergesort* algorithm based on a novel multiway merge scheme. By designing a processor-efficient two-way mergesort algorithm and finding a solution to utilize the multiway merge scheme, we obtain a new *Multiway Mergesort* on the 2D ARPBS model. Although an $O(1)$-time sorting algorithm on an AROB of size $n \times n$ is proposed in [6], it only sorts n general keys. Both of our proposed optimal sorting algorithms sort n^2 items in $O(\log n)$ time.

1.2 Paper Organization

This paper is organized as follows: Section 2 gives a detailed description of the 2D ARPBS model. Section 3 introduces several new basic algorithms on a 2D ARPBS. Section 4 presents the details for the *Generalized Columnsort* and Section 5 introduces the newly designed *Multiway Mergesort*. Finally, Section 6 concludes the paper.

2 Model Descriptions

In this section, we first briefly describe the LARPBS model and then propose a novel two-dimensional extension of the LARPBS model: the 2D ARPBS model.

2.1 The LARPBS Model

The LARPBS model is a folded optical bus system with three similar waveguides, one set of fixed delays and two sets of flexible switches. To avoid confusion, we provide two different views for the LARPBS model in Fig. 1. The three waveguides are: a *message waveguide* shown in Fig. 1a, used to carry messages, a *reference waveguide* and a *select waveguide* shown in Fig. 1b, used together to carry address information. The select waveguide has conditional delay switches on the upper half of the bus, called *transmission segment*, and the reference waveguide has fixed delay loops on the lower half of the bus, called *receiving segment*. Communication on this model is carried out through sending messages and addresses on the transmission segment at the beginning of a bus cycle and receiving them from the receiving segment in the same bus cycle. There are two sets of switches connected to each processor on the bus, namely *conditional delay switches* and *reconfigurable switches*. The conditional delay switches, shown in Fig. 1b, are used for addressing purposes. Coincident pulse technique [33] is used to implement the address scheme. The reconfigurable switches, $RST(i)$ and $RSR(i)$ in Fig. 1a, are used to segment one bus into several subbuses or connect multiple subbuses into one. Each subbus can independently work as an LARPBS. The switch setting in Fig. 1a configures the bus into two subbuses. The first subbus consists of processors P_0 and P_1. The second subbus consists of processors P_2, P_3, P_4, and P_5. Readers are encouraged to refer to [5] for more details about the LARPBS model.

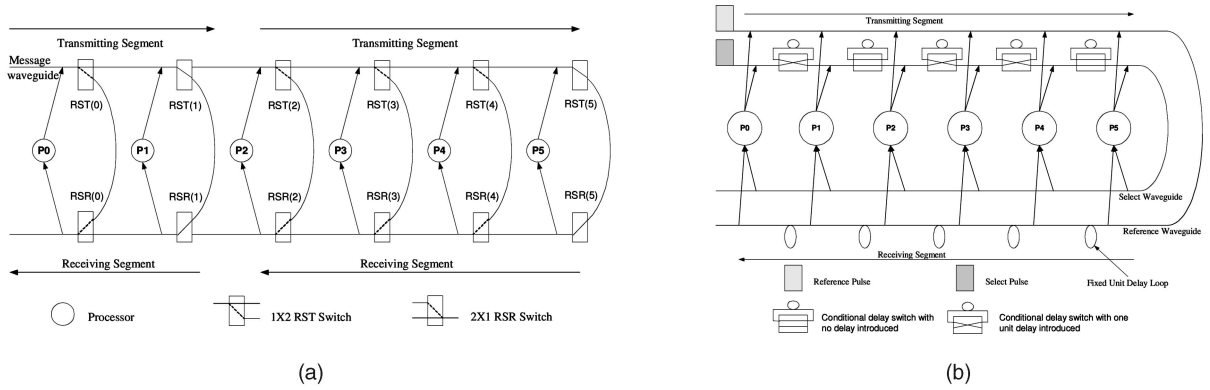

Fig. 1. The LARPBS model: (a) An LARPBS model with reconfigurable switches and (b) an LARPBS model with conditional delay switches.

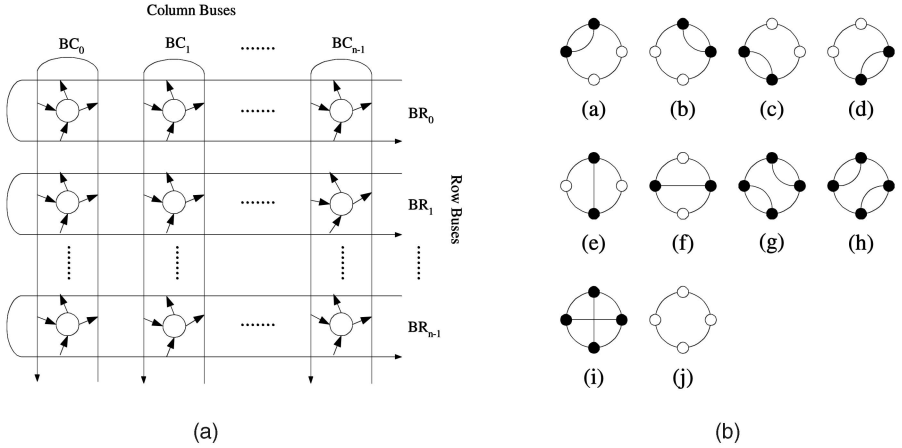

Fig. 2. A 2D ARPBS: (a) Simplified 2D ARPBS and (b) 10 possible switch settings.

2.2 Proposed Simplified 2D ARPBS

In this section, we propose a two-dimensional extension of the LARPBS model as shown in Fig. 2: the 2D ARPBS model. There are two sets of switches on the 2D ARPBS model: the reconfigurable switches and the conditional delay switches. Fig. 2 only shows reconfigurable switch connections to avoid confusion. Conditional delay switch connections on a 2D ARPBS model are the same as those of an LARPBS model. There are 10 possible switch settings [34] shown in Fig. 2b. Only the horizontal and vertical switch settings that correspond to (e) and (f) in Fig. 2b, respectively, are allowed on a 2D ARPBS model. Connection (e) can be implemented by setting the column switches to straight; connection (f) can be implemented by setting the row switches to straight. A row or a column bus can be split or combined with other buses the same way as that in the LARPBS model. A similar restricted two-dimensional model called FR-Mesh is proposed in [35]. The FR-Mesh and the 2D ARPBS are similar in that both allow only a restricted set of internal connections. The major difference between the two models is that the FR-Mesh is not an optical interconnection model. Another difference is that: in the FR-Mesh, it is possible to connect a processor to its column bus and its row bus in the same bus cycle; in the 2D ARPBS, a processor can be connected to its row bus or column bus, but not both, during one bus cycle. We choose the 2D ARPBS because its reduced hardware and communication complexity help us focus on the design of the algorithms. Another restricted two-dimensional optical interconnection model is the 2D ACPOB. Compared to the 2D ACPOB, a 2D ARPBS has one extra pair of switches, i.e., the reconfigurable switches, for easy reconfiguration.

When we extend the LARPBS model to its two-dimensional extension, the 2D ARPBS model, one of the challenges we are facing is that communication on a 2D ARPBS model is more restrictive. For example, all one-to-one communication operations on the LARPBS can be done in constant time, which does not hold any more on a 2D ARPBS. In the next section, we introduce some basic algorithms that will be used as building blocks in our sorting algorithms.

3 Basic Algorithms on a 2D ARPBS

The following algorithms were first presented by researchers in [5], [11], [26], [36]:

- *Pipelined mergesort*: It sorts n items in $O(\log n)$ time on an n-processor LARPBS model.
- *2D block BPS* (binary prefix sums): It computes binary prefix sums of binary values stored in any $x \times y$ subarray of an $n \times n$ 2D array in $O(1)$ time in column-major or row-major order.
- *2D IPS* (integer prefix sums): It computes prefix sums of $\sqrt{n}$ integers in the range of $[0, n^2 - 1]$ on a $\sqrt{n} \times n$ 2D array in $O(1)$ time.
- *2D broadcast*: It broadcasts a data item to all processors in a two-dimensional array in $O(1)$ time.

- *One-to-one communication on a row/column*: For two groups of processors $G_1 = (P_{i_0}, P_{i_1}, \ldots, P_{i_{m-1}})$ and $G_2 = (P_{j_0}, P_{j_1}, \ldots, P_{j_{m-1}})$, each processor in G_1 sends one message to one of the processors in G_2. No two processors in G_1 send messages to the same processor in G_2.
- *Broadcast on a row/column*: One processor P_i in an n-processor bus sends a message to all other $n-1$ processors $P_0, P_1, \ldots, P_{i-1}, P_{i+1}, \ldots, P_{n-1}$.
- *Matrix transpose*: This operation picks up items in each row of a matrix and aligns them in the corresponding column in the same order.
- *Reverse matrix transpose*: This operation picks up items in each column of a matrix and aligns them in the corresponding row in the same order.

All the above algorithms, excluding *Pipelined Mergesort*, run in constant time. Apart from the above known algorithms, we also design the following basic algorithms that will be introduced in details in the rest of this section: *Extended Matrix Transpose*, *Extended Reverse Matrix Transpose*, *Basic Column Sort*. For all the algorithms proposed in this paper, the input and output data are distributed to a 2D ARPBS one item per processor unless a difference distribution is mentioned. The above data distribution also holds for each step of most algorithms.

3.1 Extended Matrix Transposes

We provide two extended matrix transpose algorithms for an $r \times s$ shape matrix A with $r = ks$, where $k > 1$. To *split* matrix A into k $s \times s$ submatrices or *combine* the k $s \times s$ submatrices into one matrix A, each processor $P_{ms,j}$, where $0 \le m < k$ and $0 \le j < s$, sets its column reconfigurable switches to cross or straight. Next, we introduce the *Extended Matrix Transpose* algorithm and the *Extended Reverse Matrix Transpose* algorithm. In this paper, we use "/" for integer division and "%" for modular computation.

Algorithm 1. Extended Matrix Transpose
Input: rs items distributed to an $r \times s$ 2D ARPBS A.
Output: the transpose of A distributed to the same $r \times s$ 2D ARPBS.
Begin
1. Each processor $P_{i,j}$ sends its item to processor $P_{i',j}$, where $0 \le i < r$, $0 \le j < s$, and $i' = (i\%k)s + i/k$. Then, *split* matrix A into k $s \times s$ submatrices.
2. Apply a *Matrix Transpose* operation on each submatrix A_i, where $0 \le i < k$. Then, *combine* all submatrices into one $r \times s$ matrix.

End

Lemma 1. *For any $r \times s$ matrix A, where $r = ks$, there is a constant time matrix transpose algorithm that transposes matrix A to a same shape matrix A' on a 2D ARPBS model.*

Proof. We need to prove that item $a_{i,j}$ in matrix A is transposed to location (i', j') in A', where $0 \le i < r$, $0 \le j < s$, and

$$i' = (i\%k)s + j, \quad j' = i/k. \tag{1}$$

After step 1, item $a_{i,j}$ is moved to location (l, j), where $l = (i\%k)s + i/k$. After step 2, item $a_{l,j}$ is moved to location (i', j'), where $i' = j + l - (l\%s)$, $j' = l\%s$. By applying the equation for l from step 1 to equation for i' and j' obtained in step 2 and simplifying it, we obtain (1). This completes the proof of the lemma since each step in this algorithm takes constant time. □

Algorithm 2. Extended Reverse Matrix Transpose
Input: rs items distributed to an $r \times s$ 2D ARPBS A.
Output: The reverse transpose of A distributed to the same $r \times s$ 2D ARPBS.
Begin
1. *Split* matrix A into k $s \times s$ submatrices, and then apply *Reverse Matrix Transpose* operation to each submatrix.
2. *Combine* all submatrices into one $r \times s$ matrix, and then each processor $P_{i,j}$ sends its item to processor $P_{i',j}$, where $i' = k(i\%s) + i/s$, $0 \le i < r$, and $0 \le j < s$.

End

Lemma 2. *For any $r \times s$ matrix A, where $r = ks$, there is a constant time reverse matrix transpose algorithm that reverse transposes matrix A to a same shape matrix A' on a 2D ARPBS model.*

Proof. We need to show that item $a_{i,j}$ in matrix A is reverse transposed to location (i', j') in A', where $0 \le i < r$, $0 \le j < s$, and

$$i' = kj + i/s, \quad j' = i\%s. \tag{2}$$

After step 1, item $a_{i,j}$ is relocated to (l, j'), where $l = j + i - (i\%s)$, $j' = i\%s$. After step 2, item $a_{l,j'}$ is sent to location (i', j'), where $i' = k(l\%s) + l/s$. By replacing l with the value from step 1 and simplifying it, we obtain $i' = kj + i/s$. This completes the proof of the lemma since each of the three steps is completed in constant time. □

3.2 Basic Columnsort

In this section, we present an $O(\log r)$-time Columnsort algorithm, which is based on Leighton's seven-phase Columnsort [37]. Here are the details.

Algorithm 3. Basic Columnsort
Input: rs items distributed to an $r \times s$ 2D ARPBS A, where $r \ge s^2$.
Output: The rs input items distributed in column major order to the $r \times s$ 2D ARPBS.
Begin
1. Sort each column in ascending order using *Pipelined Merge Sort*.
2. Transpose matrix A using *Extended Reverse Matrix Transpose*.
3. Sort each column in ascending order using *Pipelined Merge Sort*.
4. Reverse transpose matrix A using *Extended Matrix Transpose*.
5. Sort each even indexed column in ascending order and odd indexed column in descending order using *Pipelined Merge Sort*.
6. Do two steps of odd-even transposition operations on each row simultaneously.
7. Sort each column in ascending order using *Pipelined Merge Sort*.

End

We now prove the correctness of this algorithm and show that it runs in $O(\log r)$ time.

Theorem 1. *There is a Columnsort algorithm on an $r \times s$ 2D ARPBS, where $r \geq s^2$, that sorts rs items in $O(\log r)$ time.*

Proof. The correctness of the *Extended Reverse Matrix Transpose* used in Phase 2 and the correctness of the *Extended Matrix Transpose* used in Phase 4 follow immediately from Lemmas 2 and 1, respectively. All phases of Algorithm 3 are based on the corresponding phases of Leighton's Columnsort algorithm. Hence, the correctness of Algorithm 3 follows directly from [37]. The odd-even transposition operation in Phase 6 can be done in constant time by two one-to-one row communication steps between adjacent processors. By Lemmas 1 and 2, the matrix transposition operations also take constant time. Since *Pipelined Mergesort* algorithm sorts r items in $O(\log r)$ time, Algorithm 3 runs in $O(\log r)$ time. □

4 A Novel Generalized Columnsort Algorithm

In this section, we present a generalized Columnsort algorithm that sorts n^2 items on an $n \times n$ 2D ARPBS in $O(\log n)$ time. Assume that $\sqrt{n}$ is an integer. The basic idea is first, partition matrix A into $\sqrt{n}$ submatrices: A_0, $A_1, \ldots, A_{\sqrt{n}-1}$, each being an $n \times \sqrt{n}$ matrix; then, virtually construct a matrix B of size $n\sqrt{n} \times \sqrt{n}$ by "stretching" all items in each submatrix into one "super column" that has $n\sqrt{n}$ items, and apply *Basic Columnsort* to B. Thus, by introducing the concept of "super column" and "virtual matrix," we convert the problem of sorting an $n \times n$ matrix to the problem of sorting an $n\sqrt{n} \times \sqrt{n}$ matrix. To separate or connect super columns, processors on the last column of each submatrix A_i set their row reconfigurable switches to cross or straight.

To provide convenience for the operations applied to the "virtual matrix," two different representations for the virtual matrix B are used. In the first representation, the ith $n \times \sqrt{n}$ submatrix A_i is used to represent the ith column in virtual matrix B. The mapping between matrix A and virtual matrix B is: $a_{i,j}$ in A is mapped to $b_{i',j'}$ in B, where $i' = i + (j\%\sqrt{n})n$, $j' = j/\sqrt{n}$, or $i = i'\%n, j = j'\sqrt{n} + i'/n$.

In the second representation, the $(m\sqrt{n} + j)$th column in matrix A is used to represent the jth column in matrix B, where $0 \leq m < \sqrt{n}$. The mapping between matrices A and B is: $a_{i,j}$ in A is mapped to $b_{i',j'}$ in B, where

$$i' = (j/\sqrt{n})n + i, \quad j' = j\%\sqrt{n}, \tag{3}$$

$$i = i'\%n, \quad j = j' + (i'/n)\sqrt{n}. \tag{4}$$

The first representation gives us the convenience of sorting each column in virtual matrix B on a submatrix A_i. The second representation makes it easy for us to perform odd-even transposition and matrix transpose operations to virtual matrix B on matrix A. To convert from one representation to the other, we design a constant time operation—*$\sqrt{n}$-way Column Shuffle*: Processor $P_{i,j}$ sends its item to processor $P_{i,k}$, where $k = (j\%\sqrt{n})\sqrt{n} + j/\sqrt{n}$, and $0 \leq i, j < n$. We would like to point out that *$\sqrt{n}$-way Column Shuffle* of a *$\sqrt{n}$-way Column Shuffle of matrix A* returns matrix A. Assume that item $a_{i,j}$ in A is moved to $a_{i,k}$ after applying *$\sqrt{n}$-way Column Shuffle* to matrix A and is further moved to $a_{i,h}$ after applying *$\sqrt{n}$-way Column Shuffle* a second time. We have the following equation $h = (k\%\sqrt{n})\sqrt{n} + k/\sqrt{n} = (j/\sqrt{n})\sqrt{n} + j\%\sqrt{n} = j$.

Now we are ready to present our generalized columnsort algorithm.

Algorithm 4. Generalized Columnsort
Input: n^2 unsorted items.
Output: A sorted sequence of the n^2 input items.
Begin
1. Separate super columns and sort them simultaneously using *Basic Columnsort*.
2. *Extended Reverse Matrix Transpose* each submatrix simultaneously, connect super columns, and then *$\sqrt{n}$-way Column Shuffle* matrix A.
3. Repeat step 1.
4. Connect super columns, *$\sqrt{n}$-way Column Shuffle* matrix A, separate super columns, and then *Extended Matrix Transpose* each submatrix.
5. Sort each submatrix simultaneously using *Basic Columnsort* with even indexed submatrices in increasing order and odd indexed submatrices in decreasing order.
6. Connect super columns, *$\sqrt{n}$-way Column Shuffle* matrix A, and then carry out two steps of *Odd-Even Transposition* operations.
7. Sort super columns: *$\sqrt{n}$-way Column Shuffle* matrix A, repeat step 1, and connect super columns to form the sorted matrix A'.

End

The correctness of this algorithm is based on the correctness of Leighton's seven-phase Columnsort algorithm. Since virtual matrix B satisfies $r \geq s^2$, we can apply Leighton's algorithm to B. It is obvious that phases 1, 3, 5, 6, and 7 in Algorithm 4 match the corresponding phases in Leighton's algorithm. To show that phases 2 and 4 also match the corresponding phases in Leighton's algorithm, we need to introduce the following two lemmas:

Lemma 3. *Extended Reverse Matrix Transpose all submatrices A_i simultaneously, where $0 \leq i < \sqrt{n}$, is equivalent to $\sqrt{n}$-way Column Shuffle matrix A followed by Extended Reverse Matrix Transpose matrix B.*

Proof. Let $a^k_{x,y}$ denote the item at location (x, y) in submatrix A_k, where $0 \leq k < \sqrt{n}$, and $a_{i,j}$ denote the same item at location (i, j) in matrix A. Then, we have

$$a_{i,j} = a^k_{i,j\%\sqrt{n}} \quad or \quad a^k_{x,y} = a_{x,y+k\sqrt{n}.} \tag{5}$$

First, we consider the first sequence of operations that includes only extended matrix transposition all submatrices A_k. According to (2), after *Extended Reverse Matrix Transpose* each submatrix A_k in A, $a^k_{x,y}$ is moved to location (x', y') in A_k, where

$$x' = \sqrt{n}y + x/\sqrt{n}, \quad y' = x\%\sqrt{n}. \tag{6}$$

Assume that location (i', j') in matrix A corresponds to location (x', y') in submatrix A_k and location (i, j) in

matrix A corresponds to location (x, y) in submatrix A_k. We obtain (7) and (8) by applying (5):

$$x = i, \quad y = j\%\sqrt{n}, \tag{7}$$

$$i' = x', \quad j' = y' + (j/\sqrt{n})\sqrt{n}. \tag{8}$$

We obtain (9) by applying (7) to (6):

$$x' = \sqrt{n}(j\%\sqrt{n}) + i/\sqrt{n}, \quad y' = i\%\sqrt{n}. \tag{9}$$

We obtain (10) by applying (9) to (8):

$$i' = \sqrt{n}(j\%\sqrt{n}) + i/\sqrt{n}, \quad j' = i\%\sqrt{n} + (j/\sqrt{n})\sqrt{n}. \tag{10}$$

Thus, the new location for $a_{i,j}$ after the first sequence of operations is given in (10). Now, we consider the second sequence of operations. After performing a *$\sqrt{n}$-way Column Shuffle* on matrix A, $a_{i,j}$ is moved to $a_{i,y}$, where

$$y = (j\%\sqrt{n})\sqrt{n} + j/\sqrt{n}. \tag{11}$$

Assume that $a_{i,y}$ is mapped to item $b_{l,h}$ in virtual matrix B, we obtain (12) using (3):

$$l = (y/\sqrt{n})n + i, \quad h = y\%\sqrt{n}. \tag{12}$$

We get (13) by applying (11) to (12) and simplifying it:

$$l = n(j\%\sqrt{n}) + i, \quad h = j/\sqrt{n}. \tag{13}$$

Assume that after *Extended Reverse Matrix Transpose* matrix B, $b_{l,h}$ is moved to $b_{l',h'}$. We can get the following equation using (2):

$$l' = nh + l/\sqrt{n}, \quad h' = l\%\sqrt{n}. \tag{14}$$

Assume that item $b_{l',h'}$ in B is mapped to item $a_{i',j'}$ in matrix A, we obtain (15) using (4):

$$i' = l'\%n, \quad j' = h' + (l'/n)\sqrt{n}. \tag{15}$$

Since we assume that $\sqrt{n}$ is an integer, we have $n/\sqrt{n} = \sqrt{n}$ and $n(j\%\sqrt{n})\%\sqrt{n} = 0$. We also notice the following facts: $j\%\sqrt{n} < \sqrt{n}$ gives us $\sqrt{n}(j\%\sqrt{n})\%n = \sqrt{n}(j\%\sqrt{n})$, $i\%\sqrt{n} < n$ gives us $i/\sqrt{n}\%n = i/\sqrt{n}$, and $0 \le l < n\sqrt{n}$ gives us $l/\sqrt{n}/n = 0$. We obtain (16) by applying (13) and (14) to (15) and simplifying it with the above facts:

$$i' = \sqrt{n}(j\%\sqrt{n}) + i/\sqrt{n}, \quad j' = i\%\sqrt{n} + (j/\sqrt{n})\sqrt{n}. \tag{16}$$

As (10) and (16) are equivalent, the lemma is proved. □

According to Lemma 3, step 2 in Algorithm 4 is equivalent to the following three operations: *$\sqrt{n}$-way Column Shuffle* matrix A, *Extended Reverse Matrix Transpose* matrix B, and then *$\sqrt{n}$-way Column Shuffle* matrix A.

Lemma 4. *Extended Matrix Transpose all submatrices A_i simultaneously, where $0 \le i < \sqrt{n}$, is equivalent to Extended Matrix Transpose matrix B followed by $\sqrt{n}$-way Column Shuffle matrix A.*

Proof. First, we consider the first sequence of operations. According to (1), after *Extended Matrix Transpose* each submatrix A_k in A, $a^k_{x,y}$ is moved to location (x', y') in A_k, where

$$x' = (x\%\sqrt{n})\sqrt{n} + y, \quad y' = x/\sqrt{n}. \tag{17}$$

Assume that location (i', j') in matrix A corresponds to location (x', y') in submatrix A_k and location (i, j) in matrix A corresponds to location (x, y) in submatrix A_k. We get (18) by applying (17) and (7) to (8) and simplifying it:

$$i' = (i\%\sqrt{n})\sqrt{n} + j\%\sqrt{n}, \quad j' = i/\sqrt{n} + (j/\sqrt{n})\sqrt{n}. \tag{18}$$

Thus, the new location for item $a_{i,j}$ after the first sequence of operations is given in (18). Next, we consider the second sequence of operations. Assume that item $a_{i,j}$ in matrix A maps to item $b_{x,y}$ in matrix B, we obtain (19) by applying (3):

$$x = (j/\sqrt{n})n + i, \quad y = j\%\sqrt{n}. \tag{19}$$

After *Extended Reverse Matrix Transpose* matrix B, $b_{x,y}$ is moved to $b_{x',y'}$. We obtain (20) by applying (1):

$$x' = (x\%n)\sqrt{n} + y, \quad y' = x/n. \tag{20}$$

Assume that item $b_{x',y'}$ in B is mapped to item $a_{l,h}$ in matrix A. We obtain (21) by applying (4), replacing x' and y' with (20), and replacing x and y with (19):

$$l = i\sqrt{n}\%n + j\%\sqrt{n}, \quad h = j/\sqrt{n} + (i/\sqrt{n})\sqrt{n}. \tag{21}$$

Assume that after $\sqrt{n}$-way Column Shuffle matrix A, item $a_{l,h}$ is moved to location (i', j'). We obtain (22) from the definition of *$\sqrt{n}$-way Column Shuffle* and (21):

$$i' = i\sqrt{n}\%n + j\%\sqrt{n}, \quad j' = (j/\sqrt{n})\sqrt{n} + i/\sqrt{n}. \tag{22}$$

A comparison of (18) and (22) shows that we still need to prove that $i\sqrt{n}\%n = (i\%\sqrt{n})\sqrt{n}$. Since $0 \le i < n$, replace i with $i = k\sqrt{n} + x$ in both expressions $i\sqrt{n}\%n$ and $(i\%\sqrt{n})\sqrt{n}$, where $0 \le k,\ x < \sqrt{n}$. We find that $i\sqrt{n}\%n = (i\%\sqrt{n})\sqrt{n} = x\sqrt{n}$. Thus, (18) and (22) are equivalent; so this completes the proof of the lemma. □

According to Lemma 4, step 4 in Algorithm 4 is equivalent to the following three operations: *$\sqrt{n}$-way Column Shuffle* matrix A, *Extended Reverse Matrix Transpose* matrix B, and then *$\sqrt{n}$-way Column Shuffle* matrix A.

Based on Lemmas 3 and 4, step 2 reverse transposes matrix B and step 4 transposes matrix B. Thus, Algorithm 4 correctly sorts the n^2 items. As all basic operations in this algorithm run in constant time and *Basic Columnsort* runs in $O(\log n)$ time, we obtain the following theorem:

Theorem 2. *There is a Generalized Columnsort algorithm that sorts n^2 items on an $n \times n$ 2D ARPBS in $O(\log n)$ time.*

5 Multiway Mergesort Algorithm

In this section, we present another optimal sorting algorithm on the 2D ARPBS model. This algorithm is based on a novel multiway merge scheme. We first present a two-way mergesort algorithm and then use it as a building block to design a multiway merge algorithm, which, in turn, will lead to our optimal multiway mergesort algorithm.

5.1 Two-Way Mergesort Algorithm

Our two-way mergesort algorithm relies on the following lemma presented in [38] with proof:

Lemma 5. *Assume that* $A = (a_1, a_2, \ldots, a_n)$ *and* $B = (b_1, b_2, \ldots, b_n)$ *are two sorted sequences, and* $a_{\lceil n/2 \rceil} \leq b_{\lceil n/2 \rceil + 1}$. *Let* $D = (d_1, d_2, \ldots, d_n)$ *be the sorted sequence obtained by merging* $b_1, b_2, \ldots, b_{\lceil n/2 \rceil}$ *and* $a_{\lceil n/2 \rceil + 1}, a_{\lceil n/2 \rceil + 2}, \ldots, a_n$. *Then, no item in sequence* $E = (a_1, a_2, \ldots, a_{\lceil n/2 \rceil}, d_1, d_2, \ldots, d_{\lfloor n/2 \rfloor})$ *is strictly larger than any item in sequence* $F = (b_{\lceil n/2 \rceil + 1}, b_{\lceil n/2 \rceil + 2}, \ldots, b_n, d_{\lfloor n/2 \rfloor + 1}, d_{\lfloor n/2 \rfloor + 2}, \ldots, d_n)$.

Based on the above lemma, we now present a two-way mergesort algorithm.

Algorithm 5. 2Way_Mergesort

Input: Two $r \times s$ matrices A and B, where $r = s^2$. Items for both matrices are distributed to an $r \times s$ 2D ARPBS, one item per processor for each matrix, i.e., processor $P_{i,j}$ holds item $a_{i,j}$ from A and item $b_{i,j}$ from B.

Output: $2rs$ sorted items are stored in the 2D ARPBS in the following column-major order $(a_{0,0}, a_{1,0}, \ldots, a_{r-1,0}, \ldots, a_{0,s-1}, a_{1,s-1}, \ldots, a_{r-1,s-1}, b_{0,0}, b_{1,0}, \ldots, b_{r-1,0}, \ldots, b_{0,s-1}, b_{1,s-1}, \ldots, b_{r-1,s-1})$, two items per processor, i.e., processor $P_{i,j}$ holds item $a_{i,j}$ and item $b_{i,j}$:

Begin

1. *Double sort*: Apply *Basic Columnsort* twice to sort matrix A and B separately.
2. *Compare medium items and form matrix D*.
3. *Sort matrix D*: Apply *Basic Columnsort* to the 2D ARPBS to sort *matrix D*.
4. *Construct matrices E and F*.
5. *Sort E and F*: Apply *Basic Columnsort* to E and F, then coalesce *E and F*.

End

Details of steps 2 and 4 in Algorithm 5 are given below.

Step 2: Processor $P_{\alpha,\beta}$, where $\alpha = (sr/2)\%r$ and $\beta = (sr/2)/r$, compares its a with processor $P_{\alpha+1,\beta}$'s b item. If $a_{\alpha,\beta} > b_{\alpha+1,\beta}$, each processor in the 2D ARPBS swaps its a and b values. Then processors in the first half of the matrix set their b values to be active and processors in the second half of the matrix set their a values to be active. All active items form matrix D.

Step 4: Each processor in the first half of the 2D ARPBS exchanges its b with a of a processor in the second half of the 2D ARPBS. See Fig. 3. All a items form matrix E and all b items form matrix F.

The following theorem shows the correctness of Algorithm 5:

Theorem 3. *The 2Way_Mergesort algorithm on an* $n \times \sqrt{n}$ *2D ARPBS sorts* $2n\sqrt{n}$ *items in* $O(\log n)$ *time.*

Proof. Based on Lemma 1, after step 4, all items in matrix A are no larger than any item in matrix B. Thus, after step 5, A followed by B is sorted in column-major order. Since the most time-consuming operation in this algorithm is *Basic Columnsort*, which runs in $O(\log n)$ time, Algorithm 5 runs in $O(\log n)$ time. □

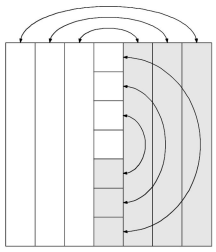

Fig. 3. Step 4 in Algorithm 5: Construct E and F. Shade areas send item a and blank areas send item b.

5.2 Multiway Merge Algorithm

The multiway merge problem (MMP) is defined as follows: Given a collection of sorted arrays $A = \{A_0, A_1, \ldots, A_{m-1}\}$, we obtain a sequence of sorted items that contains all the items in A. We are particularly interested in the MMP with $m = \sqrt{n}$ and each array has $n\sqrt{n}$ items. Our algorithm is based on a novel multiway merge scheme [38] described as follows: Let $A = \{A_0, A_1, \ldots, A_{\sqrt{n}-1}\}$ be an $n \times n$ matrix. Each A_i is an $n \times \sqrt{n}$ matrix sorted in column-major order. We take n items as samples from A, one item per column, sort these samples and use them to partition items in A into $\sqrt{n}$ subsets $B_0, B_1, \ldots, B_{\sqrt{n}-1}$ such that any item in B_i is no larger than any item in B_{i+1}, and then sort matrix A by sorting each B_i simultaneously followed by coalescing the sorted B_is. Next, we provide some definitions and properties of the merge scheme.

We define the *leader* of a column in A as the largest item in that column. Let $S = \{s_0, s_1, \ldots, s_{n-1}\}$, where $s_i \leq s_{i+1}$, be the sorted *sample set* obtained by picking the leaders of all columns in A and sorting them. Let item b be the leader of column k in A and r_b be the rank of b in the sorted sample set S. Column k is said to be *regular* w.r.t. B_i if $i\sqrt{n} < r_k \leq (i+1)\sqrt{n}$. By the ordering of S, it is easy to see that there are exactly $\sqrt{n}$ columns in A that are regular w.r.t. each B_i.

Let $C = \{C_0, C_1, \ldots, C_{\sqrt{n}-1}\}$ be $\sqrt{n}$ $n \times \sqrt{n}$ matrices such that C_i contains all columns in A that are regular w.r.t. B_i. We call an item e in C_i a *regular item* if it satisfies one of the following two conditions of the *item qualification rule*: 1) $s_{i\sqrt{n}} < e \leq s_{(i+1)\sqrt{n}}$ whenever $s_{i\sqrt{n}} < s_{(i+1)\sqrt{n}}$ and 2) $s_{i\sqrt{n}} \leq e \leq s_{(i+1)\sqrt{n}}$ whenever $s_{i\sqrt{n}} = s_{(i+1)\sqrt{n}}$. We call $(s_{i\sqrt{n}}, s_{(i+1)\sqrt{n}}]$ the *qualification interval* for B_i and denote it by I_{B_i}. We assign all regular items in C_i to regular matrix X_i. This assignment is represented by a set $X = \{X_0, X_1, \ldots, X_{\sqrt{n}-1}\}$ of regular matrices constructed from C_is as follows: if an item $c_i(x, y)$ in C_i satisfies the item qualification rule, then $x_i(x, y) := c_i(x, y)$; else $x_i(x, y) := \infty$.

A column k with leader s_b in A_i is said to be *special* w.r.t. B_t if, with s_a standing for the leader of the preceding column in A_i (if any), we have $a \leq (t+1)\sqrt{n} < b$. Let the columns with leaders s_a and s_b be regular w.r.t. $B_{j'}$ and B_j, respectively, such that $j' < j$. By the definition of a special column, column k with leader s_b is special w.r.t. $B_{j'}, B_{j'+1}, \ldots, B_{j-1}$. Assume that a column k in matrix A is special w.r.t. $B_{j'}, B_{j'+1}, \ldots, B_j$, item x in column k is not a regular item, and i is the smallest index, where $j' \leq i \leq j$, such that x satisfies the item qualification rule. We assign x to an $n \times \sqrt{n}$ matrix Y_i. All Y_is form a set $Y = \{Y_0, Y_1, \ldots, Y_{\sqrt{n}-1}\}$ of $\sqrt{n}$ $n \times \sqrt{n}$ matrices,

called *special matrices*. We fill the unassigned locations in each Y_i with ∞ items.

Now we can construct partition $B = \{B_0, B_1, \ldots, B_{\sqrt{n}-1}\}$: $B_i = X_i \cup Y_i$, where $0 \le i < \sqrt{n}$. Next, we present the following two lemmas that will be used to prove the correctness of our algorithm:

Lemma 6. *Special matrices $Y = \{Y_0, Y_1, \ldots, Y_{\sqrt{n}-1}\}$ contain all elements of A that are not in the regular matrices $X = \{X_0, X_1, \ldots, X_{\sqrt{n}-1}\}$.*

Proof. Assume that there is an item e in matrix A that does not belong to either X nor Y. We further assume that item e belongs to column k in submatrix A_i and s_a and s_b are the leaders for column $k-1$ and column k in submatrix A_i, respectively. The variables a and b are the ranks in S for the two leaders s_a and s_b. Since submatrix A_i is sorted, we have $s_a \le e \le s_b$. Based on the definition for a regular item given above, we can deduct that s_a is a regular item w.r.t. regular matrix B_j and s_b is a regular item w.r.t. regular matrix $B_{j'}$, where $j' = a/\sqrt{n}$ and $j = b/\sqrt{n}$. In other words, we have the following equation:

$$s_{j\sqrt{n}} \le s_a \le e \le s_b \le s_{(j+1)\sqrt{n}.} \tag{23}$$

Based on the definition of a special column, column k with leader s_b is special w.r.t. $B_{j'}, B_{j'+1}, \ldots, B_{j-1}$. If we assume that item e is not a special item w.r.t. any submatrix B_i, then we have $e < s_{j\sqrt{n}}$ or $e > s_{(j+1)\sqrt{n}}$, which is conflict with (23). Thus, the lemma is proved. □

Lemma 7. *Each A_i contains at most one special column w.r.t. B_j, where $0 \le i, j \le \sqrt{n} - 1$.*

Proof. Assume that a sorted sequence A_i contains two special columns w.r.t. B_j, i.e., the gth column and the kth column. We denote the rank for the leaders of these two columns by r_g and r_k, respectively. Assume that $r_g < r_k$. Based on the definition of a special column, we have $r_{g-1} \le (j+1)\sqrt{n} < r_g$ and $r_{k-1} \le (j+1)\sqrt{n} < r_k$. As $r_g < r_k$, we have $r_g \le r_{k-1}$. Thus, we obtain a contradiction $(j+1)\sqrt{n} < r_g \le (j+1)\sqrt{n}$. So the assumption cannot be true, proving the lemma. □

Lemma 8. *$B = \{B_0, B_1, \ldots, B_{\sqrt{n}-1}\}$ is a partition of items in A such that no item in B_i is strictly larger than any item in B_j for $0 \le i < j < \sqrt{n}$.*

Proof. We can derive from Lemma 7 that Y_i can be assigned at most $n\sqrt{n}$ qualified items. By the definition of X_i and Y_i, we deduce that every item in A is in exactly one of X_is or Y_is and any non-∞ item in X_i and Y_i is not strictly larger than any non-∞ item in X_{i+1} and Y_{i+1}. Thus, by assigning the non-∞ items in X_i and Y_i to B_i, we obtain a partition of items in A such that no item in B_i is strictly larger than any item in B_j for $0 \le i < j < \sqrt{n}$. This proves the lemma. □

Now we are ready to present our multiway merge algorithm. We use an $n \times n$ 2D ARPBS to merge $\sqrt{n}$ sorted sequences $A = \{A_0, A_1, \ldots, A_{\sqrt{n}-1}\}$. The matrices A, X, Y, and A' are distributed to the 2D ARPBS, one item per processor for each matrix.

Algorithm 6. Multiway Merge
Input: $\sqrt{n}$ sorted sequences $A = \{A_0, A_1, \ldots, A_{\sqrt{n}-1}\}$.
Output: A sorted matrix A'.

Begin

Step 1: Apply *Pipelined Mergesort* on the last row to sort the row leaders. Then, each row leader sends its new item and rank back to its original location, resets its item to be the received one, and broadcasts the received rank in its current column.

Step 2: Construct regular matrices X_is by moving each column to its regular matrix, then calculating qualification intervals and applying qualification rules to determine regular items.

Step 3: Construct special matrices Y_is by first identifying special columns and then sending special items to their special matrices to determine qualified special items.

Step 4: Apply *Basic Columnsort* on each submatrix to sort X_i and Y_i separately, then apply *2way_Mergesort* to merge X_i and Y_i.

Step 5: Calculate final address for each item in X_i and Y_i: First calculate number of items in each X_i *and* Y_i and integer prefix sums for each submatrix, and then calculate final destination address for each item.

Step 6: Relocate items in matrix X_i and Y_i using their final address: First use a column one-to-one communication operation to send an item to the correct row, and then use a row one-to-one communication operation to send an item to the correct column.

end

The details of steps 2, 3, and 5 are given below. Step 2 consists of the following loop:

for all $P_{i,j}$ $0 \le i, j < n, 0 \le j' < \sqrt{n} - 1$ and r is the rank of column j''s leader, **do in parallel**

$P_{i,j}$ sends its item $a_{i,j}$ to $P_{i,r}$. After receiving the item, $P_{i,r}$ sets $x_{i,j} := a_{i,j}$. Then, $P_{n-1,(j'+1)\sqrt{n}-1}$ sends $x_{n-1,(j'+1)\sqrt{n}-1}$ to $P_{n-1,(j'+2)\sqrt{n}-1}$ who then sets qualification interval for $B_{j'+1}$ to be $I := (x_{n-1,(j'+1)\sqrt{n}-1}, x_{n-1,(j'+2)\sqrt{n}-1})$ and *2D Broadcast* I in $A_{j'+1}$. $P_{n-1,\sqrt{n}-1}$ broadcasts $I = (x_{0,0}, x_{n-1,\sqrt{n}-1}]$ in A_0. $P_{i,j}$ checks $x_{i,j}$: if $x_{i,j}$ falls out of I, sets $x_{i,j} = \infty$; otherwise, informs processor $P_{i,k}$ to set $a_{i,k} := \infty$, where k is the column where $x_{i,j}$ comes from.

endfor

Step 3 consists of the following loop:

for all $P_{n-1,i'}$, $0 \le i, j, i' < n$, $i' \ne k\sqrt{n} - 1$, and $0 < k \le \sqrt{n}$, **do in parallel**

$P_{n-1,i'}$ sends its rank r to the leader on its right. Then $P_{n-1,i}$ sets r' to be the rank received ($r' = 0$ if no rank is received), calculates $u = \lceil r'/\sqrt{n} \rceil$ and $\delta = \lceil (r - r')/\sqrt{n} \rceil$, and broadcasts u and δ on its column bus. (Here, u is the smallest index of special matrices w.r.t. which the column is special and δ is the number of special matrices starting from Y_u). If $P_{i,j}$ receives special matrix information in the broadcast and $a_{i,j} \ne \infty$, it multicasts $a_{i,j}$ to the lth column of matrices $A_u, A_{u+1}, \ldots, A_{u+\delta-1}$, where $l = j/\sqrt{n}$. Finally, $P_{i,j}$ initializes $y_{i,j} := \infty$ and performs the following two operations: 1) sets $y_{i,j}$ to be the item received and 2) informs $P_{i,t}$, where $t = j + m\sqrt{n}$, $t < n$ and $m \ge 1$, to set $y_{i,t} := \infty$.

endfor

Step 5 consists of the following three substeps:

1. **for** all X_k and Y_k, $0 \le k < \sqrt{n}$, **do in parallel**
   Apply *2D Block BPS* in X_ks and Y_ks to compute the

number of non-∞ items. $|B_k|$ is obtained by adding these two numbers. Next, $P_{n-1,(k+1)\sqrt{n}-1}$ sends $|B_k|$ to $P_{k,(k+1)\sqrt{n}-1}$ who passes $|B_k|$ to $P_{k,n}$.
endfor

2. Apply *2D IPS* on the first $\sqrt{n} \times n$ submatrix to calculate the integer prefix sums PS_is for $|B_i|$s, where $PS_i = |B_0| + |B_1| + \cdots + |B_i|$ and $0 \leq i < \sqrt{n}$. PS_i is stored in $P_{i,n}$.
3. **for** all $P_{k,n}$ and $P_{i,j}$, $0 \leq k < \sqrt{n}$, and $0 \leq i, j < n$, **do in parallel**
   $P_{k,n}$ sends PS_k to processor $P_{k,(k+1)\sqrt{n}-1}$, which then *2D Broadcast* PS_k in its $n \times \sqrt{n}$ submatrix. Then, $P_{i,j}$ obtains the final address of its item by adding its rank in local submatrix $B_{j/\sqrt{n}}$ with the integer prefix sum it receives.
   endfor

Theorem 5. Multiway Merge *algorithm merges $\sqrt{n}$ sorted lists of size $n\sqrt{n}$ into one sorted list in $O(\log n)$ time on an $n \times n$ 2D ARPBS.*

Proof. It is easy to verify that steps 1-3 construct regular matrices X_is and special matrices Y_is correctly. Combining X_is and Y_is, we obtain a partition $B = \{B_0, B_1, \ldots, B_{\sqrt{n}-1}\}$. According to Lemma 6, partition B contains all the items in matrix A. According to Lemma 7, each processor in the system will receive at most one item in the multiple row multicast in step 3. Step 4 sorts B_is with each processor having exactly two items. Steps 5 and 6 distribute the non-∞ items in B_is to the $n \times n$ 2D ARPBS so that each processor has exactly one non-∞ item. According to Lemma 8, all items are in correct order after step 6. Since every step takes no more than $O(\log n)$ time, the claim is proved. □

5.3 Multiway Mergesort Algorithm

In this section, we present an optimal multiway mergesort algorithm.

Algorithm 7. Multiway Mergesort
Input: n^2 unsorted items.
Output: n^2 sorted input items.
Begin
Step 1: Reconfigure the $n \times n$ 2D ARPBS into $\sqrt{n}$ $n \times \sqrt{n}$ submatrices and run *Basic Columnsort* on each submatrix simultaneously.
Step 2: *Multiway Merge* the $\sqrt{n}$ sorted submatrices.
End

Theorem 6. *There is a Multiway Mergesort algorithm that sorts n^2 items on an $n \times n$ 2D ARPBS that runs in $O(\log n)$ time.*

Proof. By Theorems 1 and 5, both Basic Columnsort and Multiway Merge clearly sort the given items and run in O(log n) time, proving the claim. □

6 Conclusions

In this paper, we proposed a new two-dimensional optical interconnection model called 2D ARPBS and proved the effectiveness of this model by designing two new optimal sorting algorithms for this model. The first sorting algorithm generalized the idea of Leighton's seven-phase columnsort algorithm by introducing the concept of "*Virtual Matrix*" and "*Super Column.*" While Leighton's algorithm only sorts an $r \times s$ array, where $r \geq s^2$, our new columnsort algorithm can sort an $n \times n$ regular shape 2D ARPBS. The second optimal sorting algorithm is a multiway mergesort algorithm based on a novel multiway merge scheme. By designing a processor-efficient two-way mergesort algorithm and finding a solution to utilize the multiway merge scheme, we obtain a new *Multiway Mergesort* on the 2D ARPBS model. The *Parallel Mergesort* algorithm plays an important role in both of our proposed novel sorting algorithms. Since *Parallel Mergesort* runs in $O(\log n)$ time and is optimal, using it as a building block, we obtain two optimal sorting algorithms for the 2D ARPBS model.

References

[1] S. Sahni, "Models and Algorithms for Optical and Optoelectronic Parallel Computers," *Int'l J. Foundations of Computer Science,* vol. 12, no. 3, pp. 249-264, 2001.

[2] C. Qiao and R. Melhem, "Time-Division Optical Communications in Multiprocessor Arrays," *IEEE Trans. Computers,* vol. 42, no. 5, pp. 577-590, May 1993.

[3] Y. Ben-Asher, D. Peleg, R. Ramaswami, and A. Schuster, "The Power of Reconfiguration," *J. Parallel and Distributed Computing,* vol. 13, no. 2, pp. 139-153, 1991.

[4] R. Miller, V.K. Prasanna-Kumar, D. Reisis, and Q.F. Stout, "Meshes with Reconfigurable Buses," *IEEE Trans. Computers,* vol. 42, pp. 678-692, 1993.

[5] Y. Pan and K. Li, "Linear Array with Reconfigurable Pipelined Bus System—Concepts and Applications," *J. Information Science,* vol. 106, nos. 3/4, pp. 237-258, 1998.

[6] S. Rajasekaran and S. Sahni, "Sorting, Selection, and Routing on the Array with Reconfigurable Optical Buses," *IEEE Trans. Parallel and Distributed Systems,* vol. 8, no. 11, pp. 1123-1132, Nov. 1997.

[7] Y. Pan, K.Q. Li, and S.Q. Zheng, "Fast Nearest Neighbor Algorithms on a Linear Array with a Reconfigurable Pipelined Bus System," *Parallel Algorithms and Applications,* vol. 13, pp. 1-25, 1998.

[8] Y. Pan and M. Hamdi, "Quicksort on a Linear Array with a Reconfigurable Pipelined Bus System," *Proc. Second Int'l Symp. Parallel Architectures, Algorithms, and Networks,* pp. 313-319, 1996.

[9] Y. Pan, M. Hamdi, and K. Li, "Efficient and Scalable Quicksort on a Linear Array with a Reconfigurable Pipelined Bus System," *Future Generation Computer Systems,* vol. 13, pp. 501-513, 1998.

[10] A. Datta, S. Soundaralakshmi, and R. Owens, "Fast Sorting Algorithms on a Linear Array with a Reconfigurable Pipelined Bus System," *IEEE Trans. Parallel and Distributed Systems,* vol. 13, no. 3, pp. 212-222, Mar. 2002.

[11] M. He and S.Q. Zheng, "An Optimal Sorting Algorithm on a Linear Array with Reconfigurable Pipelined Bus System," *Proc. 15th ISCA Int'l Conf. Parallel and Distributed Computing Systems,* pp. 386-391, 2002.

[12] A.G. Bourgeois and J.L. Trahan, "Fault Tolerant Algorithms for a Linear Array with a Reconfigurable Pipelined Bus System," *Parallel Algorithms Appl.,* vol. 18, no. 3, pp. 139-153, 2003.

[13] A. Datta and S. Soundaralakshmi, "Fast and Scalable Algorithms for the Euclidean Distance Transform on a Linear Array with a Reconfigurable Pipelined Bus System," *J. Parallel and Distributed Computing,* vol. 64, no. 3, pp. 360-369, 2004.

[14] A.G. Bourgeois, Y. Pan, and S.K. Prasad, "Constant Time Fault Tolerant Algorithms for a Linear Array with a Reconfigurable Pipelined Bus System," *J. Parallel and Distributed Computing,* vol. 65, no. 3, pp. 374-381, 2005.

[15] S. Babvey, A.G. Bourgeois, J.A. Fernandez-Zepeda, and S.W. Mclaughlin, "A Parallel Implementation of the Message-Passing Decoder of LDPC Codes Using a Reconfigurable Optical Model," *Proc. Sixth Int'l Conf. Software Eng., Artificial Intelligence, Networking and Parallel/Distributed Computing and First Int'l Workshop Self-Assembling Wireless Networks,* pp. 288-293, 2005.

[16] M. Arock and R. Ponalagusamy, "Parallel Algorithms for Robot Path Planning with Simpler VLSI Architecture," *Int'l J. Computer Applications in Technology,* vol. 26, no. 3. pp. 157-163, 2006.
[17] D. Semé and S. Youlou, "Repetitions Detection on a Linear Array with Reconfigurable Pipelined Bus System," *Int'l J. Parallel, Emergent, and Distributed Systems,* vol. 22, no. 3, pp. 173-183, 2007.
[18] Y.R. Wang, "An Efficient O(1) Time 3D All Nearest Neighbor Algorithm from Image Processing Perspective," *J. Parallel and Distributed Computing,* vol. 67, no. 10, pp. 1082-1091, 2007.
[19] L. Chen, Y. Pan, and X.H. Xu, "Scalable and Efficient Parallel Algorithms for Euclidean Distance Transform on the LARPBS Model," *IEEE Trans. Parallel and Distributed Systems,* vol. 15, no. 11, pp. 975-982, Nov. 2004.
[20] J.L. Trahan, Y. Pan, R. Vaidyanathan, and A.G. Bourgeois, "Scalable Basic Algorithms on a Linear Array with a Reconfigurable Pipelined Bus System," *Proc. Int'l Conf. Parallel and Distributed Computing Systems,* pp. 564-569, 1997.
[21] J.L. Trahan, A.G. Bourgeois, Y. Pan, and R. Vaidyanathan, "Optimally Scaling Permutation Routing on Reconfigurable Linear Arrays with Optical Buses," *J. Parallel and Distributed Computing,* vol. 60, no. 9, pp. 1125-1136, 2000.
[22] B.J. D'Auriol and R. Molakaseema, "A Parameterized Linear Array with a Reconfigurable Pipelined Bus System: LARPBS(p)," *The Computer J.,* vol. 48, no. 1, pp. 115-125, 2005.
[23] R. Vaidyanathan, J.L. Trahan, and C. Lu, "Degree of Scalability: Scalable Reconfigurable Mesh Algorithms for Multiple Addition and Matrix-Vector Multiplication," *Parallel Computing,* vol. 29, no. 1, pp. 95-109, 2003.
[24] D.M. Chiarulli, S.P. Levitan, R.G. Melhem, M. Bidnurkar, R. Ditmore, G. Gravenstreter, Z. Guo, C. Qiao, M. Sakr, and J.P. Teza, "Optoelectronic Buses for High-Performance Computing," *Proc. IEEE,* vol. 82, no. 11, pp. 1701-1709, Nov. 1994.
[25] J.L. Trahan, A.G. Bourgeois, and R. Vaidyanathan, "Tighter and Broader Complexity Reconfigurable Models," *Parallel Processing Letters,* vol. 8, pp. 271-282, 1998.
[26] M.C. Pinotti and S.Q. Zheng, "Efficient Parallel Computation on a Processor Array with Pipelined TDM Optical Buses," *Proc. 12th ISCA Int'l Conf. Parallel and Distributed Computing Systems,* pp. 114-120, 1999.
[27] Y. Li, J. Tao, and S.Q. Zheng, "A Symmetric Processor Array with Synchronous Optical Buses and Switches," *Parallel Processing Letters,* vol. 8, no. 3, pp. 283-295, 1998.
[28] C. Qiao, "On Designing Communication-Intensive Algorithms for a Spanning Optical Bus-Based Array," *Parallel Processing Letters,* vol. 5, no. 3, pp. 499-511, 1995.
[29] Z. Guo, "Optically Interconnected Processor Arrays with Switching Capability," *J. Parallel and Distributed Computing,* vol. 23, pp. 314-329, 1994.
[30] M. Ajtai, J. Komlos, and E. Szemeredi, "An $O(n \log n)$ Sorting Network," *Proc. 15th Ann. ACM Symp. Theory of Computing,* pp. 1-9, 1983.
[31] M.S. Paterson, "Improved Sorting Networks with O(log$N$) Depth," *Algorithmica,* vol. 5, nos. 1-4, pp. 75-92, 1990.
[32] R. Cole, "Parallel Merge Sort," *SIAM J. Computing,* vol. 17, no. 4, pp. 1431-1442, 1988.
[33] S. Levitan, D. Chiarulli, and R. Melhem, "Coincident Pulse Techniques for Multiprocessor Parallel Computing," *Applied Optics,* vol. 29, no. 14, pp. 2024-2039, 1990.
[34] A.G. Bourgeois and J.L. Trahan, "Relating Two-Dimensional Reconfigurable Meshes with Optically Pipelined Buses," *Int'l J. Foundations of Computer Science,* vol. 11, no. 4, pp. 553-571, 2000.
[35] J.A. Fernandez-Zepeda, R. Vaidyanathan, and J.L. Trahan, "Scaling Simulation of the Fusing-Restricted Reconfigurable Mesh," *IEEE Trans. Parallel and Distributed Systems,* vol. 9, no. 9, pp. 861-871, Sept. 1998.
[36] K. Li, Y. Pan, and S.Q. Zheng, *Parallel Computing Using Optical Interconnections.* Kluwer Academic Publishers, 1998.
[37] F.T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes.* Morgan Kaufmann, 1992.
[38] S. Olariu, C. Pinotti, and S.Q. Zheng, "An Optimal Hardware-Algorithm for Sorting Using a Fixed-Size Parallel Sorting Device," *IEEE Trans. Computers,* vol. 49, no. 12, pp. 1310-1324, Dec. 2000.

Min He received the BEng degree in computer engineering from Hunan University, China, in 1988 and 1991, and the MS degree in system science and the PhD degree in computer science from Louisiana State University in 1997 and 2002, respectively. She is currently an associate professor in the Department of Computer Engineering and Computer Science at California State University at Long Beach. Her research interests include parallel algorithms and models, optical interconnection networks, embedded systems, and hardware/software codesign.

Xiaolong Wu received the BEng degree in mechanical engineering from Nanjing University of Aeronautics and Astronautics, China, in 1998, and the PhD degree in computer engineering from the University of Nevada Las Vegas in 2007. He is currently an assistant professor in the Department of Computer Engineering and Computer Science at California State University, Long Beach. His research interests include interconnection networks, computer network, cellular and wireless networks, and IC/Chip design. He has published more than 30 papers in the above areas. His research has been supported by the METRANS, CSULB SCAC, UKLEJA, UNLV GPSA, and Bally's Technologies Graduate Scholarship. He is a member of the IEEE.

Si Qing Zheng received the PhD degree from the University of California, Santa Barbara, in 1987. After being on the faculty of Louisiana State University for 11 years since 1987, he joined the University of Texas at Dallas, where he is currently a professor of computer science, computer engineering, and telecommunications engineering. His research interests include algorithms, computer architectures, networks, parallel and distributed processing, real-time and embedded systems, telecommunications, and VLSI design. He has published in these areas extensively. He is a senior member of the IEEE.

Burkhard Englert received the BS degree in mathematics from the University of Tuebingen in 1992 and the MS degree in computer science and the PhD degree in mathematics from the University of Connecticut in 2000. From 2000 to 2003, he was an adjunct assistant professor in the program in computing at the University of California Los Angeles. In 2003, he joined the Faculty of the Department of Computer Engineering and Computer Science at California State University, Long Beach, where he is now a professor. His current research interests include parallel and distributed algorithms, computer security and operations research. He is a member of the ACM and the IEEE.