ELSEVIER

Future Generation Computer Systems 13 (1997/98) 501–513

FGCS
FUTURE GENERATION COMPUTER SYSTEMS

Efficient and scalable quicksort on a linear array with a reconfigurable pipelined bus system

Yi Pan [a,*], Mounir Hamdi [b,1], Keqin Li [c,2]

[a] *Department of Computer Science, University of Dayton, Dayton, OH 45469-2160, USA*

[b] *Department of Computer Science, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong*

[c] *Department of Mathematics and Computer Science, State University of New York, New Paltz, NY 12561-2499, USA*

Received July 1995; received in revised form October 1996; accepted June 1997

Abstract

Based on the current fiber optic technology, a new computational model, called a linear array with a reconfigurable pipelined bus system (LARPBS), is proposed in this paper. A parallel quicksort algorithm is implemented on the model, and its time complexity is analyzed. For a set of N numbers, the quicksort algorithm reported in this paper runs in $O(\log_2 N)$ average time on a linear array with a reconfigurable pipelined bus system of size N. If the number of processors available is reduced to P, where $P < N$, the algorithm runs in $O((N/P)\log_2 N)$ average time and is still scalable. Besides proposing a new algorithm on the model, some basic data movement operations involved in the algorithm are discussed. We believe that these operations can be used to design other parallel algorithms on the same model. Future research in this area is also identified in this paper.

Keywords: Complexity; Optics; Parallel algorithm; Reconfigurable pipelined bus; Sorting

1. Introduction

In a multiprocessor system, processors can be connected via an interconnection network such as hypercube and mesh [12]. One drawback of these networks is that they provide limited connectivity between processors and their communication diameter (the maximum distance between processors) is proportional to the size of the system [10,23]. Hence, increasing the size of these networks does not result in a further decrease in the time complexities of most parallel algorithms running on them. The time complexities are lower bounded by the communication diameter of these networks. One way to overcome this problem is to use electronic buses for communication since

* Corresponding author. E-mail: pan@cps.udayton.edu. Supported in part by the National Science Foundation under Grants CCR-9211621 and CCR-9503882, the Air Force Avionics Laboratory of Wright Laboratory under Grant F33615-C-2218, an Ohio Board of Regents Research Challenge Grant, and an Ohio Board of Regents Investment Fund Competition Grant. Also supported by the AFOSR Summer Faculty Research Program.

[1] E-mail: hamdi@cs.ust.hk. Supported in part by the Hong Kong Research Grant Council under Grant RGC/HKUST 619/94E.

[2] E-mail: li@mcs.newpaltz.edu. Supported in part by the NASA/University Joint Venture (JOVE) in Research Program of National Aeronautics and Space Administration and the Research Foundation of State University of New York. Also supported by the 1996 NASA/ASEE Summer Faculty Fellowship Program.

0167-739X/98/$19.00
PII S0167-739X(97)00013-7

they provide direct communication between any two processors in the system [13].

Processor arrays with buses have become the focus of much interest due to recent advances in VLSI and fiber optic technology. Arrays with a global bus [4], arrays with multiple buses [13], and arrays with reconfigurable buses [18,19,22] have been proposed for efficient computations. In an array with reconfigurable buses, messages can be transmitted concurrently when the bus is partitioned into many segments, and the diameter problem in a point-to-point network disappears when all segments are reconfigured as a single global bus. Many different models have been proposed and many efficient algorithms have been implemented on such models. However, when there is a large amount of message transfer between different sections of the network, the bus segments themselves become a potential bottleneck.

Fiber optic communications offer a combination of high bandwidth and low error probability. Several researchers have proposed using optical interconnections to connect processors in a parallel computer system [2,7,16,17]. Among them, the distributed-memory SIMD (single instruction multiple data) computer with pipelined optical buses has received a lot of attention [9,14,17,20,30–32] due to its simplicity and low cost. In such a system, messages can be transmitted concurrently on a pipelined optical bus without partitioning the bus into several segments while the time delay between the furthest processors is only the end-to-end propagation delay of light over a waveguided bus. This design integrates the advantages of both optical transmission and electronic computation.

Several parallel algorithms such as the Hough transform [20], singular value decomposition [24], order statistics [21], sorting [8], and some numerical algorithms [11] have been proposed for arrays with a pipelined bus system. The preliminary work indicates that arrays with pipelined buses are very efficient for parallel computation due to the high bandwidth within a pipelined bus system. All the previous works are based on fixed configuration. However, different algorithms require different communication patterns. Some algorithms may even need different communication patterns during different phases of the same computation. Hence, introducing array reconfiguration can improve the efficiency of many algorithms. In this paper, we propose a new computational model termed linear arrays with a reconfigurable pipelined bus system (LARPBS) based on ideas of pipelined optical bus systems and processor array reconfiguration [25]. In such a model, messages can be transmitted concurrently on a bus in a pipelined fashion and a pipelined bus can be reconfigured dynamically under program control to suit communication needs. In order for a parallel algorithm to be efficient, times for both local operations and communication have to be small. In the LARPBS model, the communication time is the number of bus cycles used in an algorithm. An efficient sorting algorithm will be proposed for the LARPBS and its time complexity will be analyzed in this paper. It is shown that the time complexity of the algorithm compares favorably with those implemented on arrays with traditional reconfigurable electronic buses.

2. The LARPBS model

Before describing the LARPBS model, we first explain the communication mechanism of a pipelined bus system based on fiber optic technology. A pipelined optical bus system uses optical waveguides instead of electrical buses to transfer messages among electronic processors. The advantages of using waveguides can be seen as follows. Besides the high propagation speed of light, there are two important properties of optical signal (pulse) transmission on an optical bus: unidirectional propagation and predictable propagation delay per unit length. These two properties enable synchronized concurrent access of an optical bus in a pipelined fashion [9,17,30,31]. This, combined with the abilities of a bus structure to do efficient broadcasting or multicasting, makes the architecture suitable for many applications that involve intensive communication operations.

Fig. 1 shows an SIMD linear array in which electronic processors are connected with an optical bus. Each processor is connected to the bus with two directional couplers, one for transmitting on the upper segment and the other for receiving from the lower

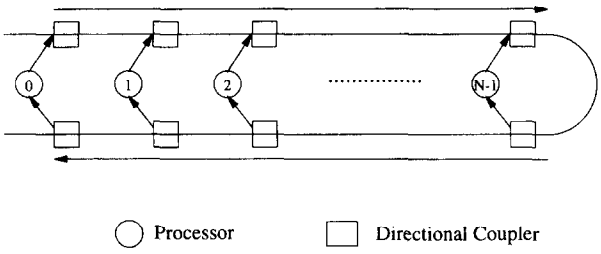


Fig. 1. A linear optical bus system of n processors.

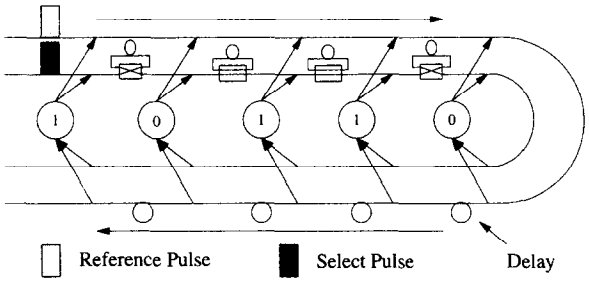


Fig. 2. An optical bus with delays added.

segment of the bus [9,17,30,31]. The optical bus contains three identical waveguides, one for carrying messages (the *message waveguides*) and the other two for carrying address information (the *reference waveguide* and the *select waveguide*), as shown in Fig. 2. For the purpose of simplicity, the message waveguide, which resembles the reference waveguide, has been omitted from the figure. Messages are organized as fixed-length *message frames*. Note that optical signals propagate unidirectionally from left to right on the upper segment and from right to left on the lower segment. This bus system is also referred to as the folded-bus connection in [9].

Although the system uses a physical line interconnect, it can simulate many communication networks such as binary trees, rings, and shuffle-exchange networks [9,17]. Thus, many complicated communication patterns can be implemented on the optical bus system. In fact, active research is being done to emulate different network topologies on the optical bus system [26,33,35].

Let ω be the pulse duration in seconds and c_b the velocity of light in these waveguides. Define a unit delay Δ to be the spatial length of a single optical pulse, i.e., $\Delta = \omega \times c_b$. Initially, processors are connected to these three waveguides such that between any two given processors, the same length of fiber is used on all three waveguides. Hence, the propagation delays between two processors are the same for all three waveguides. A bus cycle for an optical bus is defined as the end-to-end propagation delay on the bus; i.e., the time taken for an optical signal to propagate through the entire bus. If τ is the time taken for a signal to traverse the optical distance between two consecutive processors on the bus, then the length of a bus cycle for the system of Fig. 1 is $2N\tau$. We then add one unit delay Δ, shown as a loop in Fig. 2, between any two processors on the receiving segments of the reference waveguides and of the message waveguides. Each loop is an extra segment of a fiber and the amount of delay added can be accurately chosen based on the length of the segment. As a result, the propagation delays on the receiving segments of the select waveguide and the reference waveguides are no longer the same. Finally, we add a conditional delay Δ between any two processors i and $i+1$, where $0 \leqslant i \leqslant N-2$, on the transmitting segments of the select waveguides (Fig. 2). The switch between processor i and $i+1$ is called $S(i+1)$ and is local to processor $i+1$. Thus, every processor has its own switch except processor 0. The conditional delays can be implemented using 2×2 optical switches such as the $\mathrm{Ti:LiNbO_3}$ switches used in an optical computer [3]. Each switch can be set by the local processor to two different states: *straight* or *cross* as shown in Fig. 3. When a switch is set to *straight*, it takes τ time for an optical signal on the transmitting segments of the select waveguides to propagate from one processor to its nearest neighbor. When a switch is set to *cross*, a delay ω is introduced and such propagation will take $\tau + \omega$ time. Clearly, the maximum

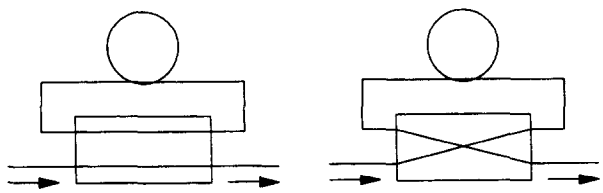


Fig. 3. Conditional delays implemented using 2×2 optical switches.

delay that the switches can introduce is the duration of $N-1$ pulses.

Messages transmitted by different processors may overlap with each other even if they propagate unidirectionally on the bus. We call these message overlappings *transmission conflicts*. Assume each message has b binary bits, each bit represented by an optical pulse, with the existence of a pulse for 1 and the absence for 0. To ensure that there are no transmission conflicts, the following condition has to be satisfied:

$$\tau > b\omega,$$

where τ is the time taken for a signal to traverse the optical distance between two consecutive processors on the bus, and ω is the pulse duration. Obviously, when b is large, τ has to be large too. This in turn means a longer bus cycle. Note that the above condition ensures that each message can fit into a pipeline cycle such that in a bus cycle, up to N messages can be transmitted by processors simultaneously without collisions on the bus. In a parallel array, messages normally have very short length; i.e., b is very small. Thus, in the following discussion, we assume that the above condition is always satisfied and that no transmission conflicts are possible as long as all processors are synchronized at the beginning of each bus cycle.

Now let us describe the LARPBS model. In the LARPBS, we insert an optical switch on each section of the transmitting bus and receiving bus. Thus, each processor has six more local switches besides its switch for conditional delay; three on its three receiving segments and three on its three transmitting segments. The switches on the receiving and transmitting segments between processors i and $i+1$ are called RSR(i) and RST(i), respectively, and are local to processor i as shown in Fig. 4. Here, RSR(i), $0 \leqslant i < N$, are 2×1 optical switches, and RST(i), $0 \leqslant i < N$, are 1×2 optical switches. In the following discussion, these switches will be called reconfigurable switches due to their function. When all switches are set to straight, the bus system operates as a regular pipelined bus system. When RSR(i) and RST(i) are set to cross, the whole bus system is split into two separate systems, one consisting of processors 0, 1, ..., and i and the other consisting of $i+1$, $i+2$,..., $N-1$. Because of the total delay for a signal passing the transmitting segment, the optical fiber between RST(i) and RSR(i), and the receiving segment is made to be equal to τ, the array with processors 0–i can operate as a regular linear array with a pipelined bus system; so does the array with processors $i+1$ to $N-1$. Fig. 4 shows the LARPBS model with six processors. The array is split into two subarrays with the first one having four processors and the second one having two processors. In the figure, only one waveguide is shown. Also, conditional switches are omitted in the figure to avoid confusion.

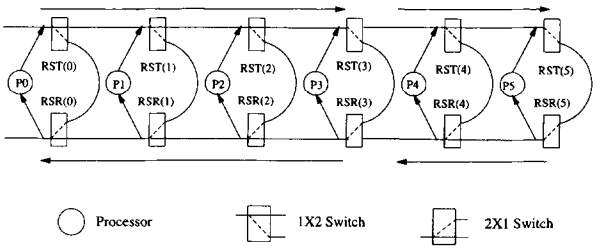


Fig. 4. The LARPBS model of size 6 with two subarrays.

Several time-division switching methods can be applied to route messages in an optical bus system. In the first approach, each processor is assigned a fixed time slot and transmits or receives a message during that particular time slot. A sequence of time slots formed on the transmitting segment of a bus is rearranged via a time-slot interchanger [34], and then forwarded to the receiving segment. Each time slot of the output sequence contains a message destined to the processor corresponding to that slot. In the second approach, each processor is assigned a fixed transmitting time slot. An SIMD environment is assumed in this case. Hence, each processor knows which processor is sending a message to it and knows the time slot that contains the message [9,17]. The last approach is to use a *coincident pulse technique* [6,8,31]. Using this approach, the relative time delay of a select pulse and a reference pulse is determined so that they will coincide, thus producing a double-height pulse, only at receiver i. By properly adjusting the detecting threshold of the detector at processor i, this double-height pulse can be detected, thereby addressing i.

In this paper, the same approach to the coincident pulse technique is used to route messages or to broadcast messages on the bus. The switches on the sending segments are used to conditionally delay the select pulses and can be closed or opened by the local processors (see Fig. 2). Now let us discuss in detail how to send a message from a source processor to a destination processor on an SIMD array using the coincidence pulse technique. First, set all switches on the transmitting segments to *straight* so that no delay is introduced in the switches on the transmitting segments of the select waveguides. A source processor sends a reference pulse at time t_{ref} (the beginning of a bus cycle) on the reference waveguide and a select pulse at time t_{sel} on the select waveguide. The source processor also sends a message frame, on the message waveguide, which propagates synchronously with the reference pulse. Whenever a processor detects a coincidence of a reference pulse and a select pulse, it reads the message frame. In other words, in order for processor i to send a message to processor j, we need to have the two pulses coincide at processor j. This happens if and only if

$$t_{sel} = t_{ref} + (N - j - 1)\omega,$$

where $0 \leqslant i, j < N$. Fig. 5 shows an address frame relative to a reference pulse for addressing processor j. Here, the select pulse is delayed for a time of $(N - j - 1)\omega$ relative to the reference pulse. Hence, the two pulses meet at processor j after the reference pulse goes through $N - j - 1$ delays.

In conclusion, for a given reference pulse transmitted at time t_{ref}, the presence of a select pulse at time $t_{ref} + (N - j - 1)\omega$ will address processor j while the absence of a select pulse at that time will not. For example, if a processor wants to send a message to processor 0, it sends a reference pulse at time t_{ref} (the beginning of a bus cycle) and a select pulse at $t_{sel} = t_{ref} + (N - 1)\omega$. Since there are $(N - 1)$ unit delays in the receiving segments of the reference waveguide, these two pulses will coincide at processor 0. In order for a processor to address processor $N - 1$, $t_{sel} = t_{ref} + (N - (N - 1) - 1)\omega = t_{ref}$; i.e., the source processor has to send a reference pulse and a select pulse at the same time. Clearly, the two pulses coincide at processor $N - 1$ since there is no delay in its receiving segment of the reference waveguide.

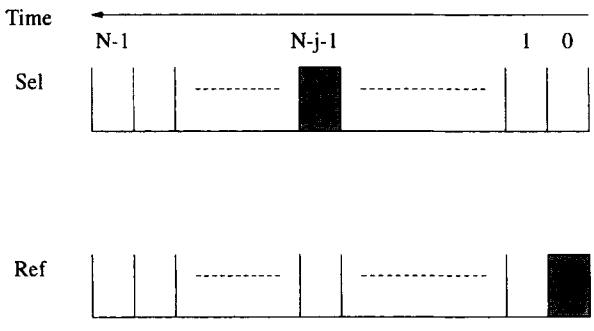


Fig. 5. Address frames.

Presumably, there are some uncertainties in the timings. A pulse traveling from one processor to the next will not take a time exactly τ. These errors may accumulate when the number of processors is large. Because the system relies on the precise timings for addressing messages, the error will eventually prohibit further scaling. This scalability problem is discussed in [5] and is called synchronization error by the authors of [5]. Some experiments have been carried out and the results indicate that large variations (of the order of one half of a pulse width) can be tolerated without significant degradation of the coincident signal [5]. For single-mode fibers with a length of a few kilometers, the synchronization error is small enough and can be tolerated. Hence, using current technology, a system using a few thousands of processors will not present any problem.

Clearly, a bus cycle is proportional to the length of the bus. However, since the optical transmission rate is much higher than the processing speed of an electronic processor, a bus cycle is of the same order of magnitude as an internal operation of an electronic processor even for a system with a few thousands of processors [9]. Many authors assume that a step is either an internal operation or a bus cycle. We adopt this method of specifying the time complexity as a number of steps. For more details on the time complexity issue, see [8,9,17,21].

In Section 3, we discuss the quicksort algorithm on the LARPBS in detail. Some basic data movement operations such as broadcast and binary summation

are discussed. Rigorous time analysis is also carried out for the algorithm.

3. The quicksort algorithm

As seen from the previous discussion, the LARPBS can be dynamically reconfigured into several LARPBSs of smaller size and each LARPBS can operate independently. Because of this property of the LARPBS model, many divide-and-conquer problems can be solved naturally and efficiently. In the following, we use quicksort as an example to show how to implement it on the LARPBS model. The purpose is to show that the LARPBS model is excellently suitable for quicksort, and other divide-and-conquer problems can also be solved efficiently in a similar fashion.

Consider a linear array with a reconfigurable pipelined optical bus of size N. The input is assumed to be a collection A of N integer numbers. We wish to sort the N elements in A. In the following presentation, we assume that the N numbers are distinct. We can assume the numbers are distinct without loss of generality since if we are given arbitrary numbers x_0, $x_1, \ldots, x_{N-1}$, we can replace x_i by (x_i, i) and define an order of the tuples by $(x_i, i) < (x_j, j)$ if $x_i < x_j$ or if $x_i = x_j$ and $i < j$.

In this paper, we present an implementation of quicksort algorithm on the LARPBS model of size N in $O(\log N)$ expected time. Our quicksort algorithm proceeds along lines similar to those in [1]. The divide-and-conquer strategy is applied to solve the sorting problem efficiently.

Suppose that C is the input array for a call to our sorting algorithm. In every iteration, we partition the set C into three disjoint subsets

$$C_1 = \{c \in C \mid c < v\},$$
$$C_2 = \{c \in C \mid c = v\},$$
$$C_3 = \{c \in C \mid c > v\},$$

where v is the pivot value of C. Since all data elements are distinct, the size of C_2 is always 1.

We move all elements in C_1 to the left-hand side of the array and move all elements in C_3 to the right-hand side of the array. Then, we divide the array into three smaller subarrays each with a size of $|C_1|$, $|C_2|$ and $|C_3|$, respectively, and apply the same algorithm to the two subarrays containing C_1 or C_3. Since the element in C_2 is larger than all elements in C_1 and smaller than all elements in C_3, it is in the right place. In this manner we can replace the given problem by two same problems of smaller size. Notice that the two problems can be solved concurrently on two subarrays. This process is continued until all subarrays have only one element left and we complete sorting set A. The whole algorithm is spelled out in algorithm *QUICKSORT*.

Algorithm QUICKSORT(D, N)

Input: A data vector D of N distinct elements are distributed in a linear array with a reconfigurable pipelined bus of size N; i.e., each processor contains a data item of the vector D.

Output: The element in the data vector D is in sorted order.

(1) In this step, we want to select a pivot number so that we can divide the current set into three subsets. Each processor in the subarray sends a reference pulse at time t_{ref} (the beginning of a bus cycle) and a select pulse at time $t_{sel} = t_{ref} + (N - i - 1)\omega$, where i is its processor index. A processor also sends a message frame containing its own index through the message waveguide. In other words, every processor tries to address itself. However, only the processor which has the largest index in the subarray will be successful in detecting a coincidence of its own reference and select pulses in its receiving segment. All other processors will fail. This can be seen as follows. The select pulse sent by the processor whose index is the largest in the subarray passes no delay in the transmitting segments. Since there are no delays in the transmitting segments and receiving segments for processor with the largest index in the subarray, the two pulses of its own will meet at the processor. For all other processors, their select pulses propagate through at least one delay in the transmitting segments and thus will not

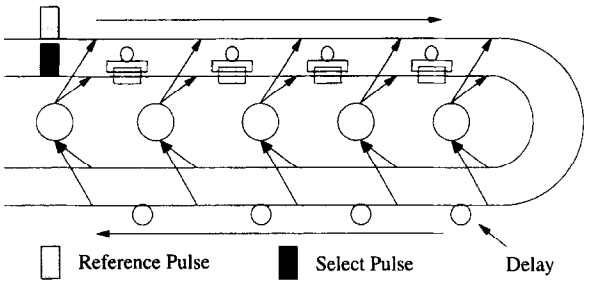


Fig. 6. Switch settings for a broadcast operation.

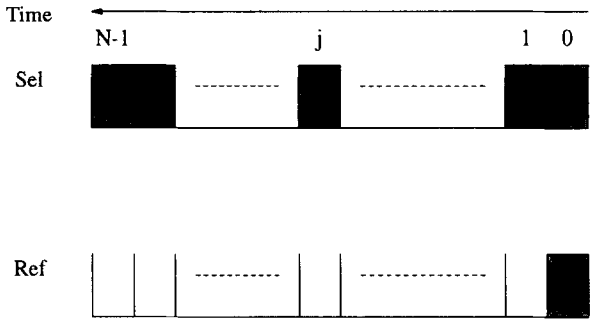


Fig. 7. Address frames for broadcasting.

coincide with their reference pulses. Thus, a processor which detects a coincidence of a reference pulse and a select pulse reads the message. If a processor receives a message containing its own index, its data item can be used as the pivot number in the next step. Otherwise, the received data item is not used in the next step.

(2) The processor which is selected in step (1) broadcasts its data item D and all processors put the received data into their local memory m, which will be used in the following steps as the pivot number. For a broadcast operation, every processor will have to detect a coincidence of the reference pulse and the select pulse in its receiving segment. This can be done as follows. All conditional switches are set to *straight*, thus introducing no delay on the transmitting segments (see Fig. 6). The source processor sends a reference pulse at the beginning of its address frame. As described before, presence or the absence of a select pulse determines if a processor should read the corresponding message or not. Thus, if the source processor sends N consecutive select pulses in its address frame on the select waveguide as shown in Fig. 7, every processor on the bus detects a double-height pulse and thus reads the message. This is clearly a broadcast operation. For example, when processor 0 in Fig. 6 wants to broadcast a message, it sends a reference pulse at the beginning of its address frame on the reference bus and five select pulses in its address frame on the select bus. The first select pulse will meet the reference pulse at processor 4 since both pulses meet no delay on their buses. The second select pulse will meet the reference pulse at processor 3 since the select pulse meets no delay on the select bus and the reference pulse goes through one delay on the reference bus. The last select pulse will meet the reference pulse at processor 0 since the select pulse meets no delay on the select bus and the reference pulse goes through four delays on the reference bus.

(3) In this step, all active processors compare the received value m with local data item D. If $m > D$, B is set to 0, indicating that the local data item is in C_1; otherwise, set B to 1, meaning that the local data item is in C_2 or C_3.

(4) Perform a binary summation over B and put the sum in s. Clearly, $s = |C_2| + |C_3|$. Now, let us explain how to perform a binary operation on a processor array with an optical bus in one bus cycle. Initially, a binary sequence $a_i = 0$ or 1 for $0 \leqslant i < N$ is distributed in the array with processor i holding a_i.

First, processor i, $1 \leqslant i \leqslant N-1$, sets its switch $S(i)$ on the transmitting segment to *straight* if $a_i = 1$, and *cross* if $a_i = 0$. Second, processor 0 injects a reference and select pulse on the reference bus and the select bus, respectively, at the beginning of a bus cycle. Note that all other processors do not put any pulse or message on the three waveguides. If processor j is selected (i.e., processor j detects the coincidence of the reference pulse and the select pulse), the sum of the $N-1$ binary numbers, a_i, for $1 \leqslant i \leqslant N-1$, is equal to j. The basic idea is to delay the select pulse whenever it passes a processor with a value of 0. When all $N-1$ processors have a value of 0, all switches on the bus

are set to *cross* and thus introduce $N-1$ unit delays. As a result, the two pulses will coincide at processor 0. When j processors have a value of 1, j switches on the bus are set to *straight* and thus introduce $(N-j-1)$ unit delays in the transmitting segments of the select waveguides. Since there are always $(N-j-1)$ unit delays on the receiving segments of the reference waveguides for processor j, the two pulses will coincide at processor j. Finally, processor j sends its index to processor 0. After obtaining $j = \sum_{i=1}^{N-1} a_i$, processor 0 gets the sum of the N numbers, a_i, for $0 \leqslant i \leqslant N-1$, by adding its local binary number a_0 to the number j.

Fig. 2 shows an example of adding five binary numbers 1, 0, 1, 1, and 0 on the LARPBS of size 5 and its corresponding conditional switch configuration. Since the binary numbers in processors 1–4 are 0110, respectively, conditional switches $S(1)$ and $S(4)$ are set to cross, and $S(2)$ and $S(3)$ are set to straight. The pulse injected from processor 0 passes through two unit delays on the select waveguide due to the conditional delays caused by $S(1)$ and $S(4)$. On the other hand, the pulse passes through two unit delays on the reference waveguide when it arrives at processor 2. Hence, processor 2 detects the coincidence of the reference pulse and the select pulse. All other processors do not detect the coincidence. After receiving a value of 2 from processor 2, processor 0 adds its local number $a_0 = 1$ to 2 and gets the sum, 3, of the five binary numbers.

(5) Calculate $|C_1| = N - s$, $|C_2| = 1$, and $|C_3| = s - 1$. Here, $|C_i|$ is the size of the set C_i for $i = 1, 2, 3$.

(6) In this step, we want to perform a split operation. Before we formally describe the data movement operation, we first introduce an operation called *compression*. When the number of active elements in the array is s, the compression algorithm will move these active elements to processors $N-s-1$, $N-s, \ldots, N-1$. In other words, the compression algorithm moves all active data items to the right-hand side of the array. Active elements are labeled based on certain value of their local variables. A processor with an active element is referred to as active processor. For example, we can label all processors with $B(i) = 1$ as active processors. In the following discussion, we assume that all active processors have their local variables $X(i)$ set to 1. The compression algorithm is implemented as follows. First, processor i sets its local switch $S(i)$ to cross if $X(i) = 1$, and to straight if $X(i) = 0$. Then, processor i whose $X(i) = 1$ injects a reference pulse at time t_{ref} (the beginning of a bus cycle) on the reference waveguide and a select pulse at time $t_{\text{sel}} = t_{\text{ref}} + N - 1$ on the select waveguide. A processor also sends a message frame containing its local data in memory location D through the message waveguide during the bus cycle. Processors with $X(i) = 0$ do not put any pulse or message on the three waveguides. In other words, every processor with an active element tries to address processor $N-1$. The select pulse sent by the processor whose index is the largest in the active set passes no delay in the transmitting segments because all the processors to its right are not in the active set and their corresponding switches are set to straight. Thus, the two pulses will meet only at processor $N-1$, and the corresponding message is picked up by processor $N-1$. Similarly, the select pulse sent by the processor whose index is the second largest in the active set passes one conditional delay in the transmitting segments because only one processor to its right is in the active set and its corresponding switch is set to cross. Since both the select and reference pulses pass one delay on the select and reference waveguides when arriving at processor $N-2$, the two pulses will meet only at processor $N-2$. Hence, processor $N-2$ receives the data item from the processor whose index is second largest in the active set. In general, the select pulse sent by the processor whose index is the kth largest in the active set passes $k-1$ conditional delays in the transmitting segments on the select waveguide because $k-1$ processors to its right are in the active set and their corresponding switches are set to cross. Since both the select and reference pulses pass $k-1$ delays on the select and

reference waveguides when arriving at processor $N-k$, the two pulses will meet only at processor $N-k$. Clearly, this is the compression operation.

Now, we describe the split operation used in this step. Specifically, we want to separate set C_1 from set C_3. In other words, all data elements $D(i), 0 \leqslant i \leqslant N-1$, whose $B(i) = 1$ are moved to the upper part of the array $\mathrm{PEN}(N-s)$, $\mathrm{PEN}(N-s+1), \ldots, \mathrm{PEN}(N-1)$, and all data elements $D(i), 0 \leqslant i \leqslant N-1$, whose $B(i) = 0$ are moved to the lower part of the array $\mathrm{PE}(0), \mathrm{PE}(1), \ldots, \mathrm{PE}(N-s-2)$, where s satisfies the following equation:

$$s = \sum_{k=0}^{N-1} B(k).$$

In other words, $D(i)$ is moved to $D(j)$ where $j = N-1-\sum_{k=i+1}^{N-1} B(k)$ if $B(i) = 1$, and $D(i)$ is moved to $D(j)$ where $j = \sum_{k=0}^{i-1} \overline{B(k)}$ if $B(i) = 0$. The split operation is performed as follows. First, we label processor i whose $B(i) = 1$ as active; i.e., set $X(i)$ to 1 iff $B(i) = 1$. We call the compression algorithm to move all data elements in the active set to the upper part of the array. To avoid destroying the original data items in D, we put them in a new memory location D_1. Second, we label all processors whose $B(i) = 0$ as active; i.e., set $X(i)$ to 1 iff $B(i) = 0$. Hence, C_1 becomes the current active set. Then, we call the compression algorithm to move all data elements in the set to the upper part of the array. To avoid destroying the data items in D and D_1, we put them in a new memory location D_2. Third, move all data items in memory location D_2 left $|C_2| + |C_3|$ positions. This is a normal data transfer operation. Using the addressing scheme discussed before, processor h, where $|C_2| + |C_3| \leqslant h \leqslant N-1$, sends its data item to processor $h - (|C_2| + |C_3|)$ by transmitting a reference pulse at time t_{ref} and a select pulse at time $t_{\mathrm{ref}} + (N - (h - (|C_2| + |C_3|)) - 1)\omega$. Finally, we copy all moved data from temporary locations D_1 or D_2 to their corresponding local locations D.

(7) Now we divide the array into three subarrays. The first subarray, i.e., processors 0, 1, ..., $|C_1|$, contains the elements in C_1. The second subarray contains only processor $|C_1| + 1$. The third subarray, i.e., processors $|C_1| + |C_2|$, $|C_1| + |C_2| + 1, \ldots, N-1$, contains the elements in C_3. This is done through setting the switches at locations $|C_1| - 1$ and $|C_1|$ to cross. In this step, we also change the indices of the third subarray so that its indices also start from 0. This can be accomplished by updating the index of processor i, where $(|C_1| + 1) \leqslant i < N-1$, to $i - (|C_1| + 1)$. Finally, we update the sizes of the two subarrays by broadcasting $|C_1|$ to all processors in the first subarray and broadcasting $|C_3|$ to all processors in the third subarray.

(8) In the last step, we recursively call QUICKSORT $(D, |C_1|)$ on the LARPBS with processors 0, 1, ..., $|C_1|$ if $|C_1| \neq 1$, and QUICKSORT$(D, |C_3|)$ on the LARPBS with processors $|C_1| + |C_2|$, $|C_1| + |C_2| + 1, \ldots, N-1$ if $|C_3| \neq 1$, The algorithm stops when both C_1 and C_3 contain one element.

The correctness of the algorithm follows by a straightforward induction on the size of the data set [1]. Clearly, for each iteration a constant number of steps is needed. In the worst case, each iteration may reduce the vector size by only one. This happens when C_1 or C_3 is empty. Hence, in the worst case a total of $O(N)$ steps are needed. This is better than the sequential quicksort algorithm discussed in [1], which has a worst-case time complexity of $O(N^2)$. The expected time of our quicksort algorithm is much better than the sequential algorithm as seen below.

Before we can talk about the expected running time of an algorithm, we must agree on what the probability distribution of the inputs is. For sorting, a natural assumption, and the one we shall make, is that every permutation of the set of numbers to be sorted is equally likely to appear as an input. Under such a condition, it is well known that the expected number of iterations required by QUICKSORT is $O(\log N)$ [1]. The total expected time of the sorting algorithm is the product of the average number of iterations and the time spent in each iteration. Therefore, we have the following theorem:

Theorem 1. *The QUICKSORT algorithm sorts a data set of N elements on the LARPBS model of size N in $O(\log_2 N)$ steps on average and uses a constant amount of memory in each processor.*

In reality, the number of data elements does not always match the size of the system. In many cases, the number of data elements is much larger than the size of the system since the system size is fixed once a machine is built. Luckily, many of the operations on the LARPBS are scalable when this situation occurs. Assume that we have a data set of N elements and P processors on the LARPBS. Also assume that each processor contains N/P data elements. It has been reported that many basic operations such as broadcast and binary summation are scalable on the LARPBS model and can be executed in N/P steps [35]. Thus, we can adapt our QUICKSORT algorithm to obtain a scalable quicksort algorithm. Since the average number of iterations remains the same, and each iteration uses $O(N/P)$ steps now, we obtain the following result:

Theorem 2. *The modified QUICKSORT algorithm sorts a data set of N elements on the LARPBS model of size P in $O((N/P)\log_2 N)$ steps on average and uses $O(N/p)$ memory elements in each processor. Clearly, the algorithm is also scalable.*

4. Conclusions

In this paper, a new computational model called LARPBS is introduced. In this model, messages can be transmitted in a pipelined fashion on an optical bus system and the bus can be dynamically reconfigured into independent segments to satisfy different communication requirements during a computation. Here, a quicksort algorithm with an average of $O(\log_2 N)$ steps is designed for this model and we show that the algorithm can be executed quickly. We also show that the quicksort algorithm is scalable when the number of processors in the system is smaller than the number of data elements.

In fact, many divide-and-conquer problems can be solved efficiently using a similar scheme. We first need to partition a problem into several subproblems. Then, we map these subproblems onto the bus system and reconfigure the system into several subsystems accordingly. Now, we can solve these subproblems on the subsystems recursively until the problem is solved completely. This again shows that reconfiguration of an optical bus system is very useful when used to solve a divide-and-conquer problem.

The new model proposed in this paper is not an optical implementation of a traditional reconfigurable bus system. They are fundamentally different. For example, on the LARPBS, a binary summation can be performed in a constant number of steps while it is impossible to accomplish such task on a traditional reconfigurable bus system in a single step. We believe that many parallel algorithms can be implemented on the LARPBS model. However, pipelined bus interconnection may require us to rethink how we write parallel algorithms. Fully exploring the capabilities of an optical bus requires careful mapping of data, an efficient addressing mechanism, and a set of efficient basic data movement operations.

The LARPBS model was first proposed in a preliminary version of the paper [25]. Since then, many new results have been achieved [26,27,33]. Several basic data movement operations and some image processing problems have been implemented on the LARPBS model [26]. Some matrix operations have also been proposed using the LARPBS model [15]. Scalability analysis on the LARPBS model is discussed in [35], and the results indicate that many commonly used basic algorithms are scalable on the LARPBS model. This implies that algorithms using only these basic algorithms are also scalable.

Because the time spent in a bus cycle is proportional to the size of the bus, it is not truly a constant. To reduce the time in a bus cycle, the linear optical bus system has been extended to two-dimensional meshes and the new system is called arrays with reconfigurable optical buses (AROB) in [27,33]. Several integer sorting problems have been solved on the AROB model [28]. Matrix operations have also been studied on the same model [29]. All these results indicate

that arrays with reconfigurable pipelined buses are powerful and practical computational models. Future research include design and analysis of more basic operations on the LARPBS, study of higher-dimensional meshes with reconfigurable optical buses, and scalability analysis of these systems. We are doing research in these directions.

Acknowledgements

We would like to thank Professor L.O. Hertzberger for his professional and timely handling of our submission. Thanks also go to the two anonymous referees for their valuable comments and suggestions.

References

[1] A. Aho, J. Hopcroft and J. Ullman, *The Design and Analysis of Computer Algorithms* (Addison-Wesley, Reading, MA, 1974).

[2] M.S. Alam and M.A. Karim, Programmable optical perfect shuffle interconnection network using Fredkin gates, *Microwave and Opt. Tech. Lett.* 5 (7) 330–333.

[3] A.F. Benner, H.F. Jordan and V.P. Heuring, Digital optical computing with optically switched directional couplers, *Opt. Engrg.* 30 (12) (1991) 1936–1941.

[4] S.H. Bokhari, Finding maximum on an array processor with a global bus, *IEEE Trans. Comput.* C-32 (2) (1984) 133–139.

[5] 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* 82 (11) (1994) 1701–1709.

[6] D. Chiarulli, R. Melhem and S. Levitan, Using coincident optical pulses for parallel memory addressing, *IEEE Comput.* 20 (12) (1987) 48–58.

[7] P.W. Dowd, Wavelength division multiple access channel hypercube processor interconnection, *IEEE Trans. on Comput.* 41 (10) (1992) 1223–1241.

[8] Z. Guo, Sorting on array processors with pipelined buses, *Proc. Int. Conf. on Parallel Processing* (St. Charles, IL, August 1992) 289–292.

[9] Z. Guo, R. Melhem, R. Hall, D. Chiarulli and S. Levitan, Pipelined communication in optically interconnected arrays, *J. Parallel Distributed Comput.* 12 (3) (1991) 269–282.

[10] M. Hamdi, A class of recursive interconnection networks: Architectural characteristics and hardware cost, *IEEE Trans. Circuits and Systems-I: Fundamental Theory and Applications* 41 (12) (1994) 805–816.

[11] M. Hamdi and Y. Pan, Efficient parallel algorithms on optically interconnected arrays of processors, *IEEE Proc. – Computers and Digital Techniques* 142 (2) (1995) 87–92.

[12] K. Hwang and F. Briggs, *Computer Architecture and Parallel Processing* (McGraw-Hill, New York, 1984).

[13] V. P. Kumar and C.S. Raghavendra, Array processor with multiple broadcasting, *J. Parallel Distributed Computing* 4 (2) (1987) 173–190.

[14] S. Levitan, D. Chiarulli and R. Melhem, Coincident pulse techniques for multiprocessor interconnection structures, *Appl. Opt.* 29 (14) (1990) 2024–2039.

[15] K. Li, Y. Pan and S.Q. Zheng, Fast and processor efficient parallel matrix multiplication algorithms on a linear array with a reconfigurable pipelined bus system, Technical Report 96–004, Department of Computer Science, Louisiana State University, 1996.

[16] A. Louri, Three-dimensional optical architecture and data-parallel algorithms for massively parallel computing, *IEEE Micro* 11 (2) (1991) 24–81.

[17] R. Melhem, D. Chiarulli and S. Levitan, Space multiplexing of waveguides in optically interconnected multiprocessor systems, *Comput. J.* 32 (4) (1989) 362–369.

[18] R. Miller, V.K. Prasanna-Kumar, D. Reisis and Q.F. Stout, Meshes with reconfigurable buses, *MIT Conf. on Advanced Research in VLSI* (1988) 163–178.

[19] K. Nakano, T. Masuzawa and N. Tokura, A sub-logarithmic time sorting algorithm on a reconfigurable array, *IEICE Trans.* 74 (11) (1991) 3894–3901.

[20] Y. Pan, Hough transform on arrays with an optical bus, *Proc. 5th Int. Conf. – Parallel and Distributed Computing and Systems* (Pittsburgh, PA, October 1992) 161–166.

[21] Y. Pan, Order statistics on optically interconnected multiprocessor systems, *Proc. 1st Int. Workshop on Massively Parallel Processing Using Optical Interconnections* (Cancun, Mexico, April 1994) 162–169.

[22] Y. Pan, Order statistics on a linear array with a reconfigurable bus, *Future Generation Computer Systems* 11 (3) (1995) 321–327.

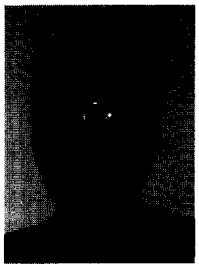
[23] Y. Pan and H.Y.H. Chuang, Properties and performance of the block shift network, *IEEE Trans. Circuits and Systems-I: Fundamental Theory and Applications* 44 (2) (1997) 93–102.

[24] Y. Pan and M. Hamdi, Singular value decomposition on processor arrays with a pipelined bus system, *J. Network and Computer Applications* 19 (3) (1996) 235–248; a preliminary version also appeared in *Proc. ACM Symp. on Applied Computing* (1993) 525–532.

[25] Y. Pan and M. Hamdi, Quicksort on a linear array with a reconfigurable pipelined bus system, *Proc. IEEE Int. Symp. on Parallel Architectures, Algorithms, and Networks* (June 1996) 313–319.

[26] Y. Pan and K. Li, Linear array with a reconfigurable pipelined bus system: Concepts and applications, *Int. conf. on Parallel and Distributed Processing Techniques and Applications* (Sunnyvale, CA, August 1996) 1431–1442; also to appear in: Special Issue on Parallel and Distributed Processing Techniques and Applications, *Inform. Sci.* (1997).

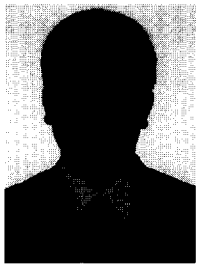
[27] S. Pavel and S.G. Akl, On the power of arrays with optical pipelined buses, *Proc. Int. Conf. on Parallel and Distributed Processing Techniques and Applications* (Sunnyvale, CA, August 1996) 1443–1454.
[28] S. Pavel and S.G. Akl, Integer sorting and routing in arrays with reconfigurable optical buses, *Proc. Int. Conf. on Parallel Processing* III (August 1996) 90–94.
[29] S. Pavel and S.G. Akl, Matrix operations using arrays with reconfigurable optical buses, *Parallel Algorithms and Applications* 11 (1996) 223–242.
[30] C. Qiao and R. Melhem, Time-division optical communications in multiprocessor arrays, *IEEE Trans. Comput.* 42 (5) (1993) 577–590.
[31] C. Qiao, R. Melhem, D. Chiarulli and S. Levitan, Optical multicasting in linear arrays, *Int. J. Opt. Comput.* 2 (1) (1991) 31–48.
[32] C. Qiao, R. Melhem, D. Chiarulli and S. Levitan, Multicasting in optical bus connected processors using coincident pulse techniques, *Proc. Int. Conf. on Parallel Processing* (August 1991).
[33] S. Rajasekaran and S. Sahni, Sorting, selection and routing on the arrays with reconfigurable optical buses, *IEEE Trans. Parallel Distributed Systems* (1997), to appear.
[34] S. Ramanan and H. Jordan, Serial array shuffle-exchange architecture for universal permutation of time slots, *SPIE Proc. Digital Optical Computing II* 1215 (1990) 330–342.
[35] 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. Conf. on Parallel and Distributed Computing Systems* (New Orleans, LA, October 1997), to appear.



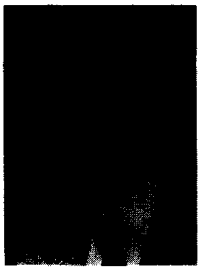
Yi Pan was born in Jiangsu, China. He entered Tsinghua University in March 1978 with the highest college entrance examination score among all 1977 high school graduates in Jiangsu Province. He received his B.Eng. degree in computer engineering from Tsinghua University, China, in 1982, and his Ph.D. degree in computer science from the University of Pittsburgh, USA, in 1991.

Dr. Pan joined the Department of Computer Science at the University of Dayton, Ohio, USA, in 1991 and has been an associate professor since 1996. His research interests include parallel algorithms and architectures, optical communication and computing, distributed computing, task scheduling, and networking. He has published more than 40 papers in international journals and conference proceedings. He has received several awards including NSF Research Opportunity Award, AFOSR Summer Faculty Fellowship, Andrew Mellon Fellowship from Mellon Foundation, and Summer Research Fellowship from the Research Council of the University of Dayton. His research has been supported by NSF, AFOSR, US Air Force, and the state of Ohio. He is the co-recipient of the Best Paper Award at the 2nd International Conference on Parallel and Distributed Processing Techniques and Applications in 1996. Dr. Pan is currently on the editorial board of the International Journal of Parallel and Distributed Systems and Networks. He will be a co-guest editor of a special issue of Parallel Processing Letters on "Computing on Bus-Based Architectures" to be published in 1998 and a special issue of Informatica on "Parallel Computing with Optical Interconnections" to be published in 1998. He will be the program vice chair of the 9th International Conference on Parallel and Distributed Computing and Systems in October 1997 and the publicity chair of the 1997 International Conference on Parallel and Distributed Processing Techniques and Applications. He has also served as a session chair or a committee member for various international conferences.

Dr. Pan is a senior member of IEEE and a member of the IEEE Computer Society. Currently, he is the Chairman of the IEEE Computer Society Student Activities Committee in Region 2 (Mideastern USA) and the Secretary of IEEE Computer Society Dayton Chapter. He is listed in Men of Achievement and Marquis Who's Who in Midwest.



Dr. Mounir Hamdi received the B.Sc. degree with distinction in Electrical Engineering from the University of Southwestern Louisiana in 1985, and M.Sc. and Ph.D. degrees in electrical engineering from the University of Pittsburgh in 1987 and 1991, respectively. While at the University of Pittsburgh, he was a Research Fellow involved with various research projects on interconnection networks, high-speed communication, parallel algorithms, switching theory, and computer vision. In 1991 he joined the Computer Science Department at Hong Kong University of Science and Technology where he is now an Associate Professor. His main areas of research are Parallel Computing. High-Speed Networks, ATM Packet Switching Architectures, and Wireless networking. Dr. Hamdi has published over 70 papers on these areas in various journals and conference proceedings. He co-founded and co-chaired the *International Workshop on High-Speed Network Computing*, is on the editorial board of the *IEEE Communications Magazine*, and has been on the Program Committee of various International Conferences. Dr. Hamdi is a member of IEEE and ACM.



Keqin Li received B.S. degree in computer science (1985) from Tsinghua University, China, and Ph.D. degree in computer science (1990) from the University of Houston. He is currently an Associate Professor of Computer Science in State University of New York (SUNY) at New Paltz. Dr. Li's research interests are mainly in design and analysis of algorithms, and parallel and distributed computing. He has published about 70 research papers on refereed journals and conference proceedings, and received best paper awards in 1996 International Conference on Parallel and Distributed Processing Techniques and Applications, and 1997 IEEE National Aerospace and Electronics Conference. His current research is extensively supported by National Aeronautics and Space Administration (NASA) and SUNY Research Foundation. Dr. Li is the associate editor-in-chief of *International Journal of Parallel and Distributed Systems and Networks*, and a guest editor of *Informatica*, and *Information Sciences – An International Journal*. He has served in various capacities for numerous international conferences, and is the program chair of 9th International Conference on Parallel and Distributed Computing and Systems (October 1997), and will be the conference chair of 4th International Conference on Computer Science and Informatics (October 1998). Dr. Li is a senior member of IEEE, and a member of IEEE Computer Society, ACM, SIGACT, SIGARCH, SIAM, ISMM, IASTED and SCS.