International Journal of Foundations of Computer Science,
© World Scientific Publishing Company

INTEGER SORTING AND ROUTING IN ARRAYS WITH RECONFIGURABLE OPTICAL BUSES

Sandy Pavel
Department of Computing and Information Science
Queen's University, Kingston, Ontario, K7L 3N6, CANADA

and

Selim G. Akl
Department of Computing and Information Science
Queen's University, Kingston, Ontario, K7L 3N6, CANADA

Received (received date)
Revised (revised date)
Communicated by (Name of Editor)

Abstract

In this paper we present deterministic algorithms for integer sorting and on-line packet routing on arrays with reconfigurable optical buses. The main objective is to identify the mechanisms specific to this type of architectures that allow us to build efficient integer sorting, partial permutation routing and h-relations algorithms. The consequences of these results on the PRAM simulation complexity are also investigated.

Keywords: Optical pipelined buses, reconfigurable array, sorting, routing.

1. Introduction

In large-scale general purpose parallel machines based on connection networks, efficient communication capabilities are essential in order to solve most of the problems of interest in a timely manner. Interprocessor communication networks are often the main bottlenecks in parallel machines. One important limitation of these networks concerns the exclusive access to the bus resources, which limits throughput to a function of the end-to-end propagation time. Optical communications have emerged as a solution to this problem. Unlike electronic buses on which signal propagation is bidirectional, optical channels are inherently unidirectional and have predictable delay per unit length. As it is shown in [11, 21], this allows a pipeline of signals to be created by the synchronized directional coupling of each signal at specific locations along the channel. Using this kind of spatial parallelism, the end-to-end propagation latency can be amortized over the number of parallel messages active at the same time on the bus. A number of arrays with optical pipelined communications are proposed in [11, 13, 21, 24, 25, 29, 31].

We introduce in [24, 25] a model which incorporates some of the advantages and characteristics of two existing models, namely the classical reconfigurable networks [20] and the arrays with optical pipelined buses. The new *Array with Reconfigurable Optical Buses*, AROB, is used in this paper as the model for which we study the possibility of implementing integer sorting and routing algorithms. When using optics, techniques which are unique and/or suitable to optics must be developed. Some of these techniques are either revised or are introduced in this paper. The aim is to identify those techniques which allow for simple and efficient algorithms.

Integer sorting is the first problem studied. The performances of the integer sorting algorithms given in this paper are immediate consequences of the particular communication and computation mechanisms used. Under certain conditions (to be specified later), sorting n integers takes $O(1)$ steps on a linear AROB (LAROB) with n processors. This is to be compared with the complexity of the algorithm given in [32] which takes $O(k)$ steps when n k-bit integers are sorted on a LAROB with the same number of processors. Note that sorting n bits on the CRCW PRAM needs $\Omega(\log n/\log\log n)$ time, for any polynomial number of processors [6]. [A PRAM consists of n processors and m memory locations, where each processor is a random-access machine, see for example [1]. All processors communicate via the shared memory. There are two cases for the write operation: exclusive write EW, when only one processor is allowed to write into a given location and concurrent write (CW) where two or more processor can write into a given location. Similarly for the read operation we distinguish between concurrent read (CR) and exclusive read (ER). When several processors attempt to write into the same arbitrary memory location of a CW variant, the model must specify what value ends up stored in that memory location. Examples of these rules, given in the decreasing order of their power, are: Priority, Common and Combining CW. Obviously, the concurrent (CW,CR) models are more powerful than the exclusive ones, (EW,ER).]

It is also shown in this paper that the RotateSort algorithm for n^2 values $a_{i,j}$, $0 \leq a_{i,j} \leq n-1$ takes $O(1)$ steps on the $n \times n$ AROB. The more general case of sorting n general values has been considered in [2, 12, 17, 24]. Of particular interest for us is the possibility of using integer sorting in order to implement efficient algorithms for different packet routing problems.

The routing problem requires each processor in a network, which initially stores one packet, to send it to its destination processor without interfering with other packets. The task is to send all packets as quickly as possible with the smallest possible *queue size*. The *queue size* is the maximum number of packets that have to be stored simultaneously in a single processor during any step of the routing algorithm. Two types of routings are studied in this paper, namely the *partial permutation routing* and the *h-relations*. Consider that each processor of a network stores at most one data packet which must be routed to an arbitrary destination processor. The *partial permutation routing* (or 1-relation) problem requires to move the packets to their destinations, each processor being the destination of at most one packet [36]. The partial permutation routing of n packets on an n-processor LAROB can be done on-line in one step [11, 24]. It is easy to prove that the same permutation routing algorithm can also be used when each processor has $O(1)$ packets

[24]. If off-line preprocessing is allowed, it is shown in [11, 24] that an arbitrary partial permutation routing on a two-dimensional array with optical pipelined buses can also be implemented in $O(1)$ steps. In [26] it is shown that the *barrel shift* permutation routing can be implemented on-line in $O(1)$ steps on the 2D AROB. The subset of permutations called *bit permute and complement* can also be implemented in $O(1)$ steps on the 2D AROB [33]. The basic computation and communication mechanisms, particular to this type of optical networks, allow us to give in this paper $O(1)$-step deterministic algorithms for on-line arbitrary partial permutation routing on the 2D and 3D AROBs.

The h-relation is the communication problem which requires that the (up to) h packets stored by each processor be moved to their arbitrary destinations, with the condition that each processor is the destination of at most h packets [36]. It has been shown in [4] that a special case of $(\log n)$-relations can be routed in $\tilde{O}(\log n)$ steps (i.e. in $O(\log n)$, with high probability) on an n-node *Optical Communication Parallel Computer* (OCPC). In an OCPC any processor can communicate with any other processor in one unit of time, provided there are no conflicts. In case of conflict, no message reaches the intended destination [4, 14, 36]. The best time of $\tilde{O}(h + \log\log n)$ has been obtained in [14] for the general OCPC h-relation problem. It is shown in [32] that: 1) the h-relations on an n-node LAROB take $\tilde{O}(h)$ steps; 2) arbitrary h-relations can be routed in $\tilde{O}(h + \log\log n)$ steps on an $\sqrt{n} \times \sqrt{n}$ AROB; and 3) there is an $O(h \log n)$-step deterministic algorithm for any h-relation on a LAROB as well as a 2D AROB. We show that, in general, for any h-relation simulated on either linear or 2D AROBs, there are on-line deterministic algorithms which take $O(h)$ steps. It takes $\Omega(n)$ steps for a processor to read/write h data and therefore the h-relation algorithms are optimal.

It has been shown in [24] that an n-processor Priority-CRCW PRAM can be simulated in $O(1)$ steps by a linear array processor with pipelined buses. Also, one step of a CRCW PRAM can be simulated on $n^{1/2} \times^{1/2}$ array processors with pipelined buses in $O(\log n)$ steps with high probability [24]. The on-line partial permutation routing and h-relations algorithms presented in this paper allow us to extend the study of PRAM simulation on the AROB. It is shown that a p-processor linear or two-dimensional AROB, where each processor has $O(n/p)$ memory, can simulate in $O(n/p)$ steps any operation performed on an n-processor EREW PRAM, with $p \leq n$. When $p = n$ the EREW PRAM can be simulated in $O(1)$ steps on the 3D AROB. As imediate consequence, one step of the n-processor Priority-CRCW PRAM can be simulated deterministically in $O(n/p \log n)$ steps on a p-processor 2D AROB, $p \leq n$, and in $O(\log n)$ steps on a 3D AROB with n processors.

2. The Model

2.1. Linear arrays with optical pipelined buses

Consider a linear array of n processors connected to an optical bus as in Fig. 1. Each processor is connected to the bus through two directional couplers. One is used to write data on the bus and the other to read the data from the bus. Each processor sends data on the upper (*transmitting*) segment of the bus and reads messages from the lower (*receiving*)

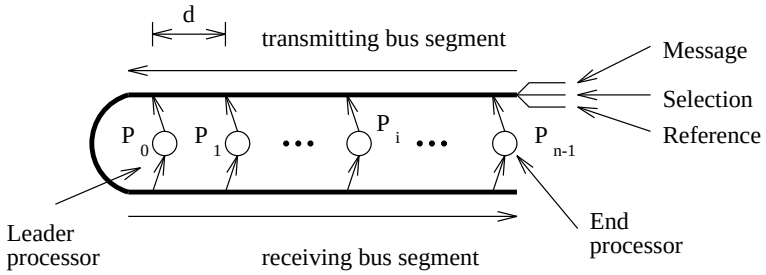

Figure 1: The linear array with pipelined optical buses APPB.

segment of the bus. The optical bus is constructed from three identical waveguides. One waveguide is used for message transmission, the *message* waveguide, and two are used for carrying address related information, the *reference* (Ref) and *select* (Sel) waveguides. During a write cycle, the data written by a processor into the bus propagate as indicated with arrows in Fig. 1, and may be read by any subsequent node on the bus. Due to the directionality of the signal propagation and the predictable delay of the signal, the same bus may be used to transmit messages between other nodes in the same time. Let us consider that each message is b bits long. Each bit is represented by an optical signal of width w seconds for a binary value of 1. The absence of such a signal represents a 0. We call the position occupied by a message a *time slot*.

Two conditions are essential for this kind of transmission: (1) all transmissions are synchronized and (2) the length of the optical path on the waveguide between any two adjacent nodes, d in Fig. 1, satisfies:

$$d > bwc_g, \tag{1}$$

where c_g is the velocity of light in the waveguide. These two conditions ensure that the signals corresponding to two different messages, which travel in a waveguide in the same direction at the same time, do not physically overlap at any point on the waveguide, i.e., are space multiplexed. The *end-to-end propagation time* is $\sigma_b = 2n\tau$ seconds, where n is the number of processors in the linear array and $\tau = d/c_g$, is the time taken for a pulse to traverse the optical distance d. A τ-time period is called a *petit cycle*. The principles of this linear *Array Processor with Pipelined Buses* (APPB) have been introduced in [11, 21].

Several approaches can be used to route messages in a linear APPB structure from one processor to another. In [11, 21] a *time waiting function* is introduced. In [29] a time-division multiplexing scheme is given for communications in arrays with optical pipelined buses. This is used in [24, 25] to implement a part of the synchronous communications needed by the AROB. The address information can also be encoded using the *coincident pulse technique* and *unit time delays* as shown in [10, 19, 29, 30]. This technique can be used when the source processor knows the address (index) of the destination processor. A *unit time delay* is represented by the duration of a pulse w and can be obtained using a waveguide (or fiber) segment of length equal to the spatial length of the optical pulse, that is

$w \times c_g$ [29]. For example, a processor can use the coincident pulse addressing mechanism in order to broadcast a value to all other processors [23]. Many other communication patterns are described in [10, 19, 30] using the same coincident pulse technique. The coincident pulse mechanism seems to avoid the traditional bottleneck which characterize most of the on-line address decoding techniques [29].

It is easy to verify that these communication mechanisms allow for optimal simulation of the OCPC on arrays with optical pipelined buses.

A *bus cycle length* is given by the time taken by the end-to-end propagation of the messages together with the time required to process a message at the source and destination processors. The latter operations include message generation and detection, synchronization, buffering, etc. [11]. The message processing is done in parallel by all source processors, at the beginning of the bus cycle, and/or by all destination processors after the message propagation ends. The ratio ρ between the message processing time and the time required for communication over a distance d, is estimated to be on the order of 10 to 1000 [11]. In order for the message processing time to be of the same order of magnitude as the end-to-end propagation time σ_b, the number of processors should satisfy $n \geq \rho$. In this case the bus cycle length can be considered to be $O(\sigma_b)$. Let a *step* be either an internal operation or a communication step. It has been argued that counting the number of steps of an algorithm simulated on an array with optical pipelined buses allows for a better comparison between algorithms, [12]. This method has the advantage that it abstracts from the details introduced by the technology dependent parameters, as σ_b for example. Another argument is given by the fact that although the signal propagation time is proportional to the length of the connection, for sufficient long wires, this is the best achievable performance [8]. This is a natural physical limitation of which we must be aware but which does not need to be included in the time complexity expression given for the AROB algorithms. We adopt in this paper the method of specifying the time complexity as a number of steps. For more details on the time complexity issue see [11, 13, 17, 21, 23, 24].

2.2. The AROB Model

Two-dimensional arrays with optical pipelined buses have been proposed in order to reduce the number of processors connected to a linear optical pipelined bus [11, 13, 31, 29]. The *Array with Reconfigurable Optical Buses* (AROB) we propose in [24, 25] uses the basic architectural and functional structure of a classical reconfigurable network, RN, [5]. Consider n^2 processors connected in a two-dimensional reconfigurable structure as depicted in Fig. 2.a. The switching system of a processor is able to connect the four ports (N, S, E, W) in one of the configurations shown in Fig. 2.b. The interconnection network is assumed to use optical waveguides and optical switches for communication and reconfiguration, respectively. The communication system of this network is modified in order to allow the implementation of the pipelined optical communication mechanisms as they are introduced in [11, 21] and also presented in this section. It is shown in [24] that the local switch setting provides a mechanism for building arbitrary APPB-like linear structures within the AROB. A processor can be connected to up to four such buses at a time. The functional and architectural characteristics of the AROB are described in more

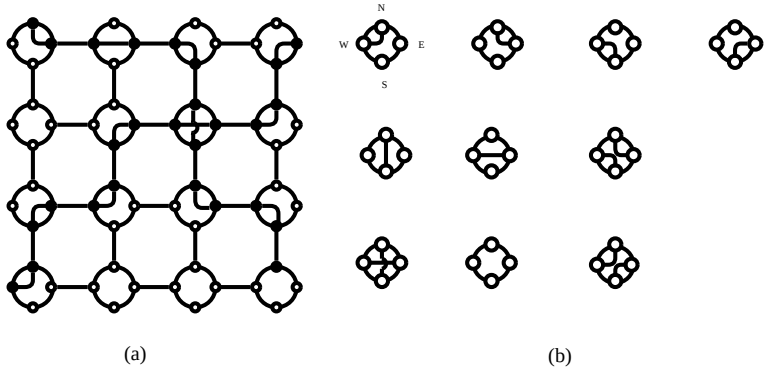

Figure 2: (a) Two-dimensional AROB; An example of possible bus configurations. (b) Allowed switch states. See text for more details.

details in [24]. An $1 \times n$ AROB is denoted as the *linear* AROB (LAROB) [24, 25]. An n-processor LAROB and the two possible switch states are depicted in Fig. 3.a and Fig. 3.b, respectively.

The description of 2D AROBs can be extended to higher dimensions $k > 2$. For example a 3D AROB processor has two extra ports for the communications with the neighbor processors in the third dimension. The switch configurations allow any two pairs of these 6 ports to be connected internally, with any port connected to only one of the other ports in the same time.

3. Basic Algorithms

3.1. Integer prefix sums

In this section we consider the special case of computing the *integer prefix sums* of n numbers v_i with $0 \leq v_i \leq n$, $0 \leq i \leq n-1$, and $\sum_{i=0}^{n-1} v_i \leq n$. Assume that an n-processor LAROB is available and that each processor p_i stores one value v_i. The prefix sums are given by the sequence $(v_0, v_0 + v_1, v_0 + v_1 + v_2, \ldots, v_0 + v_1 + \ldots + v_{n-1})$. We identify two possible techniques which can be used to find the integer prefix sums.

Technique (i): The first technique is similar to the one used in [24] to compute the binary prefix sums (BPS) of n one-bit values. In the first step each p_i introduces a delay of v_i units on the receiving segment of the selection waveguide. In the next step the leader p_0 sends synchronously a pair of light pulses, one on the reference waveguide and one on the selection waveguide. At p_i the selection pulse is delayed by v_i pulse positions relative to the reference pulse. The relative delay is decoded off-line by each p_i and represents the prefix sum $v_0 + v_1 + \ldots + v_i$. Note that as $\sum_{i=0}^{n-1} v_i \leq n$, the total delay is at most equal to n. The connection of the delay lines is done off-line in the first step. During the second step there is only one message which travels on the bus. This has maximum length

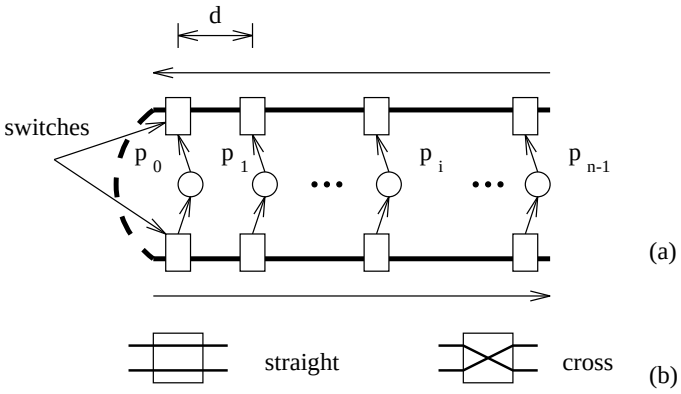

Figure 3: (a) The Linear Array with Reconfigurable Optical Buses LAROB. (b) The switch states.

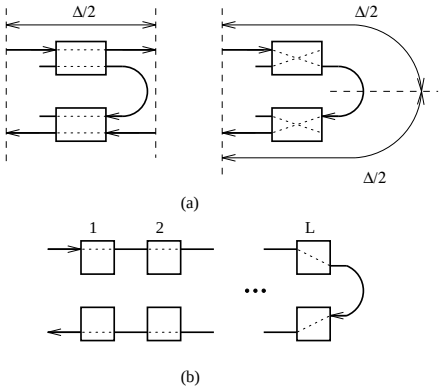

Figure 4: Programmable delays. a) A stage of one unit delay. b) The switch configuration for L Δ delays.

n and the condition imposed by equation (1) is satisfied. The programmable delays could be implemented using unit-delay (Δ) fiber segments and switches as in Fig. 4.a. When a processor has to introduce a delay of L Δ units, a number of L delay stages are coupled as in Fig. 4.b. Another possibility is to use ($\log n$) stages with the delay at stage i being 2^i. This reduces the number of switches needed to implement the programmable delay lines. Although very simple, this technique has the main drawback that each processor must have n unit delays and more importantly, a considerable number of switches are required. Alternative solutions can be found in [7] and [28], for example.

Technique (ii): For the second variant we note that at most n relative delays are required in order to find the prefix sums. This allows the unit delay to increase from w to τ, while keeping the bus cycle length in $O(\sigma_b)$. The timing mechanisms of each processor can be used for the required operations, rather than using fiber and switches

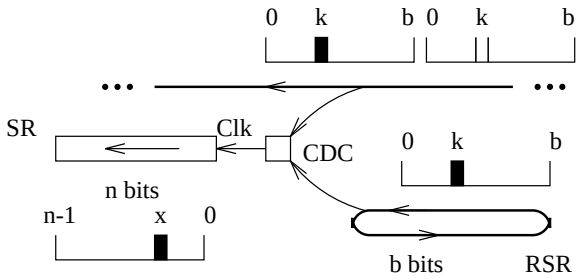

Figure 5: The bit polling mechanism.

to construct the delays. Consider a bus cycle which starts with p_0 sending a pulse on the transmitting segment of the bus. After one petit cycle, the pulse reaches p_0 on the receiving segment. Let us denote this moment as t_0. The internal timing circuit is triggered by this pulse and after v_0 petit cycles, p_0 sends a pulse to p_1. In general, if at t_k a pulse is detected at p_k, another pulse is sent on the bus after v_k petit cycles. In order for this mechanism to function properly each processor which stores a value greater than zero must interrupt the receiving segment of the bus by setting (off-line) the switch in the cross state. For the zero value no bus segmentation is performed as no delay is required. The value $((t_k - t_0)/\tau - (k-1)) = v_0 + \ldots + v_{k-1}$, computed by p_k, represents the delay, in number of petit cycles, introduced by p_i, for all i, $0 \leq i \leq k-1$. The total delay is no more than $n\tau$ and the bus cycle takes $O(\sigma_b)$. That is, the bus cycle is within the limits imposed by the model. Equation (1) is also satisfied. Therefore,

Claim 1 *The prefix sums of n integers v_i with $0 \leq v_i \leq n$, $0 \leq i \leq n-1$, and $\sum_{i=0}^{n-1} v_i \leq n$, can be computed in $O(1)$ steps on an n-processor LAROB.* ■

Observation. The second technique (ii) can be interpreted as a *modified* version of the *skewed-time clock distribution model* presented in [29].

3.2. Bit polling operation

The *bit polling* operation calls for reading (detecting) the values of the k-th bits (pulses) of all (or a subset of) the n messages which propagate on the bus during a bus cycle, with $0 \leq k \leq b$. The objective is to perform this operation during one bus cycle, i.e., with the same speed as the communication. It is assumed that each processor has an *optical rotate-shift register* RSR integrated in the communication subsystem, Fig. 5. This can easily be constructed from a waveguide (fiber) loop of length equal to the optical distance between two adjacent processors, d. Therefore, a light pulse which propagates inside the RSR loop has a period equal to a petit cycle, τ. The output of the RSR is connected to a *coincident-pulse detection circuit* CDC, which can be an adapted variant of the one described in [19], for example. The second input of the CDC is connected to the bus, Fig. 5. As it is implied by its name, the CDC generates a 1 signal at the output if and only if two light pulses reach its inputs at the same time.

For the k-th bit polling operation, the RSR is loaded with 2^k at the beginning of a bus cycle. When one b-bit message arrives at p_i a signal is generated at the output of the CDC if and only if the k-th bit of the message on the bus is 1. This information can be further processed in different ways. Due to the applications to follow, we are interested in determining the number of k-th pulses which are 1. Therefore, either a counter or a shift register can be used to accumulate this information. The choice depends on the actual technological constraints which require that the update period t_u of this device be $t_u \leq \tau$. We assume without loss of generality that the latter relation is satisfied. Two consecutive pulses with index k arrive τ time apart and their number is not larger than n. Thus,

Claim 2 *With the resources specified above, the bit polling operation can be performed in $O(1)$ steps on an n-processor LAROB.* ■

Observation. In case a shift register is used, it is assumed that the off-line decoding of the unary value stored by the shift register takes $O(1)$ time. This assumption is similar to that allowing a memory access to be counted as a unit time operation, even though the circuit depth required to decode an address is proportional to the logarithm of the memory size. However, the time requirements for the off-line operations can be relaxed to be in $O(\sigma_b)$ which is the time taken by a bus cycle. Descriptions of an optical counter can be found in [16] and of an optical shift register in [15].

4. Sorting on the LAROB

In this section we give a number of algorithms for the integer sorting problem. Essentially, there are two types of resources employed by these algorithms. The first type uses the mechanisms presented in [24, 25] for the *basic* (L)AROB. The *basic* (L)AROB model presented in [24, 25] does not use switch setting during the message propagation on the bus. An *extended* (L)AROB, i.e. which allows on-line switching, is used in the second part of this section and $O(1)$-step integer sorting algorithms are obtained. The general system design conditions which must be satisfied in this case are also identified. The on-line switch setting for communication has previously been used in [13] for example.

Variation of two different sorting techniques are presented in this paper. The first variant is a stable sorting algorithm which is based on computing for each input key the rank corresponding to its sorted order, followed by a partial permutation routing. The second technique is based on the radix-sort principle, i.e. if n numbers in range $[0..I]$ can be stably sorted using p processors in time t, then n numbers in $[0..I^{O(1)}]$ can also be stably sorted in $O(t)$ with the same number of processors. The motivation for this approach is that the rank-based stable sorting seems to use efficiently the mechanisms specific to the LAROB model when the integer keys to be sorted are restricted to values at most equal to n. The rank-based stable sorting is further used as a building block for the radix-sort algorithm which allows the range of the input keys to be increased.

4.1. Integer sorting on the (basic) LAROB

The first sorting algorithm presented is for n values v_i, with v_i represented with $(\log\log n)$ bits, i.e. $0 \leq v_i \leq \log n$ for $0 \leq i \leq n-1$. Assume that v_i, ps_i, $A(i)$ and

$B(i)$ are variables stored by p_i for all i. The architecture used is an n-processor LAROB without on-line switching and with $b > \log n$. Let $S_a = \{v_k \mid v_k = a, 0 \leq k \leq n-1\}$ and $P_a = \{p_k \mid v_k \in S_a, 0 \leq k \leq n-1\}$, for any a, $0 \leq a \leq \log n$.

Algorithm Integer Sorting 1

1. (a) On the transmitting segment, p_k writes 2^{v_k} on the bus, for all k in parallel.
   (b) On the receiving segment, p_k starts polling the v_k-th bit of the data on the bus, $0 \leq k \leq n-1$. This operation is performed until the end of the cycle, i.e. for n petit cycles, and the result is stored by p_k in $B(k)$, $0 \leq k \leq n-1$.
2. Apply the multiple binary prefix OR algorithm in [24] and for each value a, $0 \leq a \leq \log n$, mark processor i as *selected*, such that $i = \min_k\{k \mid v_k \in S_a, 0 \leq k \leq n-1\}$.
3. Each selected processor $p_i \in P_a$ performs $A(a) = B(i)$ for all a in parallel, $0 \leq a \leq \log n$.
4. Use a partial permutation routing step and exchange $B(k)$ with $B(n-k-1)$, $0 \leq k \leq n-1$.
5. Each processor p_k, $0 \leq k \leq n-1$, reads $A(v_k)$ and performs $ps_k = A(v_k) + B(k)$.
6. Apply a partial permutation routing and move v_k to p_{ps_k}, $0 \leq k \leq n-1$. □

Claim 3 *The stably-sorting of n values $0 \leq v_k \leq \log n$, $0 \leq k \leq n-1$, takes $O(1)$ steps on the n-processor LAROB.*

Proof: The value $B(k)$ obtained by p_k at the end of the first step is the number of processors p_i, with $i \leq n-k$ and $v_i = v_k$. In step 2 the processor $p_i \in P_a$ which has computed $|S_a|$ is marked as *selected*, $0 \leq a \leq \log n$. The $|S_a|$ value is stored in $A(a)$ during the third step, $0 \leq a \leq \log n$. In the fourth step the entries of the vector B are moved to their correct position in the LAROB. The rank of each element is computed in step 5 and the permutation routing in step 6 moves the elements to their sorted order. The relative positions of equal keys are given in vector B. The same relative position is also maintained in the final output sorted order. Therefore the sorting operation is stable. ■
Using the radix sort,

Corollary 1 *An n-processor LAROB can sort in a constant number of steps n integers in range $[0..(\log n)^{O(1)}]$.* ■

The result in Claim 3 can be used to give an algorithm for sorting integers in range $[0..n^{O(1)}]$ which are represented with $k = c \log n$ bits, for some constant $c \geq 1$. Without loss of generality assume that k is multiple of $(\log\log n)$. We divide the k bits in $(k/\log\log n)$ sub-groups each of $(\log\log n)$ bits. The sorting algorithm proceeds in $(k/\log\log n)$ stages. During stage i the values are stably sorted with respect to the i-th least significant group of bits. In view of Claim 3, each of the $(k/\log\log n)$ stages takes $O(1)$ steps. Therefore,

Claim 4 *A LAROB with n processors can sort n k-bit values in $O(k/\log\log n)$ steps.* ■

The result in Claim 4 compares favorably with the sorting algorithm given in [32] which takes $O(k)$ steps in the same conditions.

4.2. Integer sorting on the (extended) LAROB

One objective of this paper is to identify low level communication and computational mechanisms specific to this type of architecture and to use them into different, more general applications. *Low level* refers to the mechanisms which allow very simple logic and/or arithmetic operations to be performed on-line. Two examples of low level operations are the bit polling and binary prefix sums. The assumption used so far was that the system is designed to support the simultaneous transmission of up to n messages with b bits each. Another restriction was that no on-line switch setting is performed. We show next that, by relaxing either of these conditions, sorting n integers can be performed in $O(1)$ steps on the (*extended*) LAROB.

4.2.1. Multiple Binary Prefix Sums Algorithm

Assume that we have an n-processor LAROB storing a set S of n *one-bit* values v_i, $0 \leq i \leq n-1$, one value per processor. S is assumed to be the union of m disjoint subsets S_j, $0 \leq j \leq m$, $m \leq n-1$. The *Multiple Binary Prefix Sums* (MBPS) problem requires that the binary prefix sums (BPS) [24] of the elements in each subset S_j be computed. The order of the elements in a subset is given by the relative positions of the processors which store these values. Let $P_j = \{p_k \mid v_k \in S_j\}$, $0 \leq j \leq m$. A write cycle [24] can be used to find the binary prefix sums of the values in a subset. Let us associate with each subset S_j a work time slot s_j which is known to all processors in P_j, $0 \leq j \leq m$. The initial value stored by each s_j is zero. The BPS of the elements in S_j could be computed if each processor $p_k \in P_j$ reads s_j and then adds v_k to the value stored by s_j. The value associated with s_j, which is read by p_k during this bus cycle, is the actual binary prefix sum of the elements in S_j with indices smaller than k. The definition of the (L)AROB does not allow a processor to read a value from the bus, to use it in an internal operation and then to write it back on the bus, all in τ time. However, the updating operation needed to be performed by p_k is very simple and requires only that the value stored by s_j be incremented if $v_k = 1$. If $v_k = 0$ no operation is performed. We suggest two possible techniques specific to this type of interconnection network which can be used to implement the updating operation.

Technique (i): Consider the framework of the *coincident pulse technique* [10, 19, 24, 29, 30]. The address messages on the reference and selection waveguides have n bits. Additionally, each processor can introduce a unitary delay Δ on the selection waveguide at a well specified moment of time. The unit delay can be connected to the bus through a switch as in Fig. 6.a. At the beginning of a bus cycle each processor writes a pair of pulses on the transmitting segments of the reference and the selection waveguides, respectively. Processor $p_k \in P_j$ performs the update operation by introducing a unit delay after $wait_1(k,j) = k + j + 1$ petit cycles from the beginning of the bus cycle if $v_k = 1$. If $v_k = 0$, no delay is introduced, see Fig. 6.a. The update operation performed by p_k (if any) should affect only the value carried by s_j and thus, the switch state is reversed after another petit cycle, i.e. at $wait_2(k,j) = wait_1(k,j) + 1$. Practical optical switches have a finite switching time $t_s > 0$ which currently is on the order of hundreds of picoseconds. To ensure that the switching process does not affect the messages on the bus, the effect of the switching

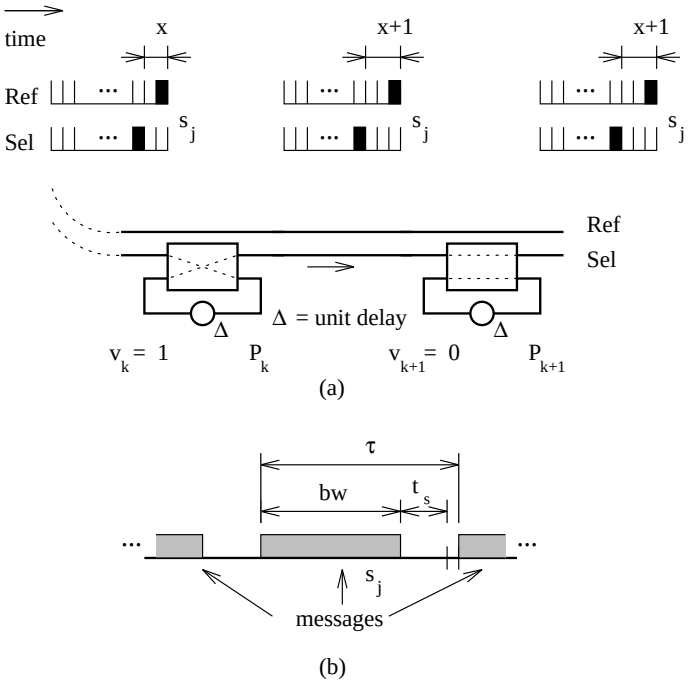

Figure 6: a) Illustration for the MBPS algorithm. b) Time diagram for the relation between τ, t_s and bw.

time must be taken into consideration. A switch must change its state before and after a message reaches that processor and the general non-overlapping condition in equation (1) becomes $bw + t_s < \tau$, see Fig. 6.b. There is no need to assume an extra t_s time gap after each message because this actually coincides with the one from the beginning of the next message. If $v_k = 1$, the effect of the update operation performed when s_j reaches p_k is to increment by one unit the selection pulse delay relative to the reference pulse. The relative position of the two pulses in the address frame read by p_k during s_j, encodes the sum of the values from the same subset S_j which are stored by processors with indices smaller than k. The disadvantage of this technique is that it requires to accommodate up to n messages on the bus, each of which has $\max\{b, n\}$ pulse slots. For big values of n this might imply long and therefore inefficient buses.

Technique (ii): The second technique presented here uses only a single n-pulse message. The n pulses are initially positioned δ time apart of each other with δ a value to be specified later, see Fig. 7.a. For simplicity and without loss of generality assume that δ is a multiple of w. Any processor can introduce a unit time delay on the selection waveguide. At the beginning of the bus cycle the switch is in the straight state and no delay is introduced.

Assume that p_k stores $v_k = 1$. During the bus cycle the message propagates on the selection waveguide from p_0 to p_{n-1}. When the j-th pulse of the message is detected at

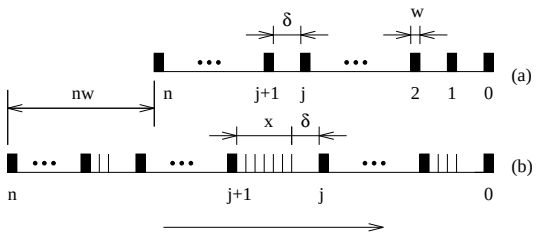

Figure 7: The second possible technique for the MBPS.

p_k, the switch state is changed and the unit delay is introduced, in the optical path of the selection waveguide, between pulses j and $j+1$ of the message. The unit delay remains active until the end of the bus cycle, therefore no other relative positions between any other pairs of adjacent pulses are affected. Maintaining the delay on the waveguide ensures also that no pulse overlapping occurs. Simultaneously with the switch setting, every processor reads the message on the bus. The number x of the actual unit delays introduced between pulses j and $j+1$ represent the number of 1 values stored by processors in S_j which have indices smaller than k. See Fig. 7.b. If $v_k = 0$ no switch setting occurs at p_k.

Besides the switching time t_s, δ should also include the time required by the mechanism used, by each processor, in order to detect the presence of the j-th pulse. Similar to the case of bit polling operation, this mechanism can be implemented using either a $(\log n)$-bit counter or an n-bit shift register. The maximum time length of the message on the bus is in this case $(n+1)(\delta + 2w)$. When $\delta + 2w \leq bwc$, for any constant c, the overall updating operation takes $O(1)$ steps. For example, typical values for the switching time and the pulse width are $t_s = 100$ picoseconds [34] and $w = 50$ picoseconds [29], respectively.

4.2.2. LAROB stable sort algorithm

Consider n values v_i, with $0 \leq v_i \leq n-1$, $0 \leq i \leq n-1$. In what follows we present an algorithm to sort these values in $O(1)$ steps on the LAROB which allows on-line switch setting. Let $S_a = \{v_i \mid v_i = a, 0 \leq i \leq n-1\}$ and $P_a = \{p_i \mid v_i \in S_a, 0 \leq i \leq n-1\}$ for any a, $0 \leq a \leq n-1$.

LAROB integer stable-sort Algorithm 2

1. Apply the MBPS algorithm and find for each v_i its relative position in S_a. The one-bit value used by each processor in the MBPS steps is 1 and any subset has no more than n elements. In order to implement the MBPS algorithm, each subset S_a has associated the work slot s_a, $0 \leq a \leq n-1$. Let r_i be the rank computed by p_i.

2. During the MBPS step the last element in S_a, i.e. p_i with $i = \max_k\{k \mid p_k \in S_a, 0 \leq k \leq n-1\}$, has computed $m_a = |S_a|$, with $0 \leq m_a \leq n$ and $\sum_{a=0}^{n-1} m_a \leq n$. The multiple binary prefix OR algorithm [24] can be used to determine in parallel the p_i with $i = \max_k\{k \mid p_k \in S_a, 0 \leq k \leq n-1\}$. Let us call these processors *active*.

3. Apply a permutation routing step during which each *active* processor in S_a sends m_a to p_a, $0 \leq a \leq n-1$.
4. Apply the integer prefix sums algorithm and determine the prefix sums ps_a of the m_a values. In another bus cycle all $p_i \in S_a$ read the prefix sum ps_a computed at p_a $0 \leq a \leq n-1$.
5. In $O(1)$ steps each processor can determine the rank of its element in the sorted sequence, i.e. $(r_i + ps_{v_i})$. As no two elements can have the same index, a permutation routing can be used in order to move the v_i's to their final positions. □

Each of these operations takes $O(1)$ steps and thus,

Claim 5 *Stably-sorting n values v_i, with $0 \leq v_i \leq n-1$, $0 \leq i \leq n-1$, takes $O(1)$ steps on an n-processor LAROB.* ■

Combining this algorithm with the radix-sort, the result can be extended to:

Corollary 2 *Sorting n integers in the range $[0..n^{O(1)}]$ can be performed in $O(1)$ steps using an n-processor LAROB.* ■

5. Partial Permutation Routing on a 2D AROB

We show in this section that the two-dimensional partial permutation routing can be performed in $O(1)$ steps on a 2D AROB. We give next a short description of a 2D integer sorting algorithm which is also used in the partial permutation routing algorithm.

The LAROB sorting algorithm can be used to construct the 2D AROB sorting algorithm. Assume we have n^2 values $a_{i,j}$, $0 \leq a_{i,j} \leq n-1$, $0 \leq i,j \leq n-1$, stored one per processor in an $n \times n$ AROB. The basic algorithm used is the RotateSort given in [22] for a generic $R \times S$ array. This uses a constant number (sixteen) of row- and column-transformation phases. Each phase consists of sorting operations on rows and/or columns and permutation routings (rotations) in rows. As an $n \times n$ AROB is used to simulate the RotateSort, there are at most n packets involved in any of these phases which are implemented in the row and column LAROBs of the 2D AROB. Each datum has a value between 0 and $n-1$. In view of Claim 5 each of these operations can be performed in $O(1)$ steps using the n-processor LAROBs in rows and columns. Thus,

Claim 6 *The RotateSort algorithm for n^2 values $a_{i,j}$, $0 \leq a_{i,j} \leq n-1$, $0 \leq i,j \leq n-1$, takes $O(1)$ steps on the $n \times n$ AROB.* ■

Consider that each processor of an $n \times n$ AROB stores at most one packet which must be routed to an arbitrary destination processor. The *partial permutation routing* (or 1-relation) problem requires to move these packets to their destinations, each processor being the destination of at most one packet [36].

Partial Permutation Routing Algorithm

1. Apply RotateSort to move the packets in the sorted, row-major order of their destination-row indices.

2. If there are exactly n^2 packets to be permuted then a permutation routing in each row will move all the packets to their destination processors. If there are processors which do not send or receive any packet then two steps are required. After step (1), the packets which have the same destination row form a contiguous segment in the row-major indexing order. No more than n packets have the same destination row, thus, no two packets in a column have the same destination row. Apply a permutation routing in each column and move all packets to their destination rows.
3. A row partial permutation routing moves all packets to their final destination processors. □

Note that a row index i satisfies $0 \leq i \leq n-1$ and that there are n^2 such indices to be sorted. Thus, phase 1 can be performed in $O(1)$ steps. Each permutation routing in phases 2 and 3 takes also $O(1)$ steps. Thus,

Claim 7 *On-line partial permutation routing can be performed in $O(1)$ steps on an $n \times n$ AROB.* ■

An immediate consequence is that:

Corollary 3 *An n-processor two-dimensional AROB can simulate in a constant number of steps any operation of an EREW PRAM with the same number of processors.* ■

6. h-relations algorithms on the (L)AROB

6.1. h-relations on the LAROB

Consider a LAROB with n processors each storing up to h packets. The h-relation is the communication problem which requires to move the up to h packets in each processor to their destinations. The destinations can be arbitrary except that each processor is the destination of at most h packets [36]. There are at most nh packets to be moved. Imagine the packets as being stored in an $h \times n$ array. Each column of this array represents a processor and each row represents a level (or memory location) on which each processor stores one packet. Therefore, the h-relation problem can be seen as a *modified* partial permutation routing in the $h \times n$ array using only an n-processor LAROB. The difference is that the actual level on the destination processor is no longer significant, as we want to move the packets to their destination processors regardless the memory location it occupies. The algorithm for the h-relation on the LAROB is a simple adaptation of the 2D partial permutation algorithm.

LAROB h-relations Algorithm

1. Simulate the RotateSort algorithm and sort the packets over their destination-column indices. During this phase the packets are moved to their positions in the column-major indexing order of the $h \times n$ array.
2. Apply the LAROB permutation routing algorithm on each level (row) in order to move all data to their destination processors. □

Claim 8 *The on-line h-relations algorithm takes $O(h)$ steps on an n-processor LAROB.*

Proof: If each processor has exactly h data, the algorithm can stop after the first phase. Otherwise, after the sorting phase (1), a sub-sequence of packets which have the same destination column, could be moved to 2 adjacent processors. After sorting no two packets with the same destination can be in the same level of the $h \times n$ array, therefore the second phase suffices in order to move the packets to their destination columns which are stored by the destination processors.

The values used by the RotateSort algorithm are the column (processor) indices, i.e. they are between 0 and $n-1$. The sorting and the permutation routing operations in each row can be performed in a constant number of steps using the entire LAROB. These operations must be performed h times and thus, the simulation of a RotateSort row operation takes $O(h)$ steps. Sorting on a column can be performed in $O(h)$ steps using a sequential integer sorting algorithm [3]. Therefore, the RotateSort simulation takes $O(h)$ steps on the LAROB. The permutation routing in the second phase takes $O(h)$ steps. ■

Of interest for practical parallel processing is the possibility to simulate efficiently, i.e. with minimum slowdown, an n-processor machine M on a p-processor machine M', with $p \leq n$. In view of Claim 8:

Corollary 4 *A p-processor LAROB, where each processor has $O(n/p)$ memory, can simulate in $O(n/p)$ steps any operation performed on an n-processor EREW PRAM, with $p \leq n$.* ■

Note that the trivial lower bound for this simulation, $\Omega(n/p)$, is given by the number of steps required by a LAROB processor in order to simulate the internal operation of the n/p associated PRAM processors. Thus the result in Corollary 4 is optimal.

6.2. h-relations on the AROB

Let us consider an $n \times n$ AROB with each processor storing at most h packets. The two-dimensional h-relation problem calls for moving these packets to their arbitrary destinations, where each processor can receive at most h packets. There are at most n^2h packets which can be seen as arranged in an $n \times n \times h$ array, with the base array $(i, j, 0)$, $0 \leq i, j \leq n-1$, representing also the AROB, see Fig. 8. The '$*$' sign is used in order to simplify the notation and represents all the values of the index it replaces in the notation.

The 2D h-relation algorithm consists of a combination of the algorithms presented so far.

2D h-relation Algorithm

(i) In each array at level k, $0 \leq k \leq h-1$, sort the packets over the indices of their destination columns. The column-major indexing is used for the destination processors.

(ii) Apply the h-relations algorithm in each array $(i, *, *)$, $0 \leq i \leq n-1$.

(iii) Apply the h-relations algorithm in each array $(*, j, *)$, $0 \leq j \leq n-1$. □

Claim 9 *The on-line 2D h-relation algorithm takes $O(h)$ steps on the $n \times n$ AROB.*

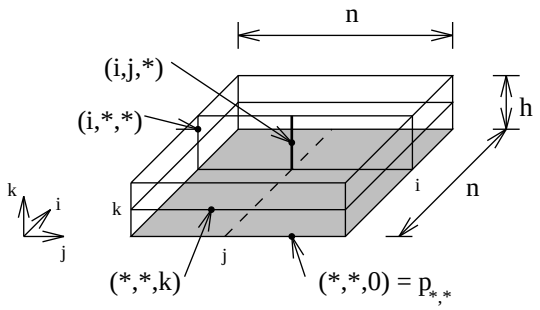

Figure 8: The two-dimensional AROB and the logical arrangement of packets in a 3D array.

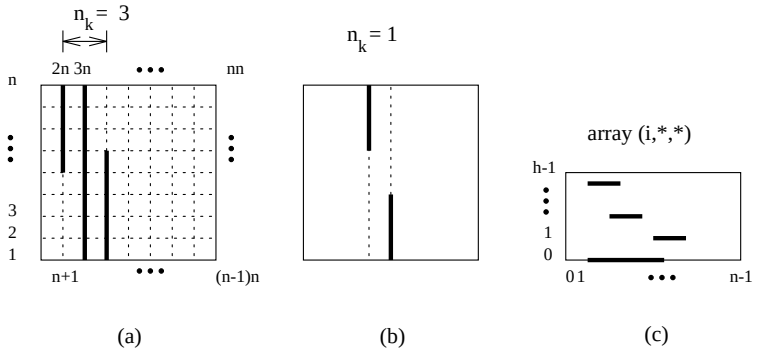

Figure 9: a),b) Illustration of the column-major indexing and of a contiguous segment of packets with the same destination column c for the array at level k. c) Possible positioning of the elements with the same destination column c in an array $(i, *, *)$.

Proof: The sorting in phase (i) takes $O(1)$ steps for each level. Thus, the overall sorting takes $O(h)$ steps. After sorting, for any array at level k, the packets which have the same destination column c - say, for some c $0 \leq c \leq n-1$, form a contiguous segment in the column-major ordering. See for example Fig. 9.a and Fig. 9.b. Note that there are at most nh packets in such a segment.

Let us denote by $n_{k,i}(c)$ the number of packets with the same destination column c which are stored in row i of the array at level k. After sorting the packets in a row i which have the same destination c form a contiguous segment. Let $n_k(c) = \max_i\{n_{k,i}(c)\}$, for any fixed k and c. For example, in Fig. 9.a $n_k(c) = 3$ and in Fig. 9.b $n_k(c) = 1$. After phase (i):

$$n_k(c) - n_{k,i}(c) \leq 1 \tag{2}$$

for any c, k and i.

Consider an $h \times n$ array $(i,*,*)$ for some i, $0 \leq i \leq n-1$, as in Fig. 9.c. At level k of this array the packets with destination column c form a contiguous segment (as explained above) of length $n_{k,i}(c) \leq n_k(c)$. The total number of packets with the same destination column c in array $(i,*,*)$ is $\sum_{k=1}^{h} n_{k,i}(c)$.

The total number of packets with the same destination column c in the entire array must be smaller than or equal to nh. That is,

$$\sum_{i=0}^{n-1}\sum_{k=0}^{h-1} n_{k,i}(c) \leq nh. \tag{3}$$

If i and c are fixed, from equation (2) we obtain:

$$\sum_{k=0}^{h-1}(n_k(c) - n_{k,i}(c)) \leq h. \tag{4}$$

That is, we have an upper bound on the difference between the number of packets when each level k of $(i,*,*)$ is assumed to have the maximum $n_k(c)$ packets (that is, in the worst case scenario) and the real number of packets. For all matrices $(i,*,*)$, for a fixed c and taking into consideration equations (3) and (4), we have:

$$\sum_{i=0}^{n-1}\sum_{k=0}^{h-1} n_k(c) \leq \sum_{i=0}^{n-1}(\sum_{k=0}^{h-1} n_{k,i}(c) + h) \leq 2nh. \tag{5}$$

The inner sum in the leftmost term does not depend on i, thus:

$$\sum_{k=0}^{h-1} n_k(c) \leq 2h. \tag{6}$$

Equivalently, if we assume for some i that after phase (i) there are $n_k(c)$ packets with the same destination column c stored at each level k of array $(i,*,*)$, then there are at most $2h$ such packets in the entire array $(i,*,*)$. This is an upper bound on the number of packets with the destination-column c which can be found after phase (i) in an array $(i,*,*)$, for any i and c.

Phases (ii) and (iii) are actually $(2h)$-relation algorithms which take $O(h)$ steps and are implemented on the associated LAROBs. Note that in order to be able to perform these two phases the processors should have queues of length $2h$. ■

Furthermore,

Corollary 5 *A p-processor 2D AROB, where each processor has $O(n/p)$ memory, can simulate in $O(n/p)$ steps any operation performed by an n-processor EREW PRAM, with $p \leq n$.* ■

It is trivial to give a partial permutation routing algorithm for a 3D AROB using the result in Claim 9. After applying the three phases of the h-relations AROB algorithm with $h = n$, an extra permutation routing is needed in each column $(i,j,*)$ and all the packets are moved to their destination processors. Thus,

Claim 10 *The partial permutation routing can be performed in $O(1)$ steps on the three-dimensional AROB. Also, any operation performed by an n-processor EREW PRAM can be simulated optimally, in $O(1)$ steps, on the n-processor 3D AROB.* ■

It is shown in [18] that one step of the Priority-CRCW PRAM with n processors can be simulated by an n-processor EREW PRAM in $O(\log n)$ steps. Therefore, one step of the n-processor Priority-CRCW PRAM can be simulated deterministically in $O(n/p \log n)$ steps on a p-processor 2D AROB, $p \leq n$, and in $O(\log n)$ steps on a 3D AROB with n processors.

7. Conclusions

We have identified in this paper a number of communication and computation mechanisms which allow us to build efficient integer sorting, partial permutation routing and h-relations algorithms on the AROB. It is shown that if on-line switch setting is allowed, sorting n integers in range $[0..n^{O(1)}]$ takes $O(1)$ steps on a LAROB with n processors. An $O(1)$-step deterministic algorithm for the partial permutation routing is also presented. In general, the h-relations algorithms designed for the (L)AROB take $O(h)$ steps.

The implications on the complexity of PRAM simulation, of the results obtained for the communication primitives, are also determined. This work was motivated by the need to relate the architectural models which use optical pipelined interconnections to computational models which have support features for high-level, general-purpose programming languages. The PRAM and the Bulk-Synchronous Parallel (BSP) computer [36, 37] are parallel computation models which allow concurrent memory access, virtual processors, barrier synchronization and automatic and explicit memory allocation, features which are essential in order to support high-level programming languages. It is well known that given enough slackness the OCPC can efficiently simulate the BSP [36, 37]. It is trivial to verify that the OCPC can be simulated optimally on an array with optical pipelined buses. Thus, the BSP can also be simulated on an array with optical pipelined buses. (We conjecture here that a more efficient and direct simulation of the BSP by the (L)AROB is possible.)

The model of computation used in this paper has electronic digital processors, fiber interconnections and opto-electronic couplers. Some of the issues not analyzed in this paper are the synchronization of processors, clock distribution, pulse positioning, optical fanout, delay lines, offline setup, etc. These problems have been investigated in [9, 27, 29]. The current opto-electronic technology is still immature and many problems remain to be solved. One example is given by the increasing optical power consumption along an optical bus, making it necessary to use optical amplifiers on buses connecting more than a few hundreds processors. The use of on-line switching is of particular importance for the AROB because signal amplitude restoration and re-synchronization can be accomplished by the technique of switching in a fresh copy of the system clock, see [16]. However, although expensive, all the opto-electronic devices assumed by our model exist. Present commercial devices limit the operating speed to a few hundreds of MHz. Demonstrations of optical transmissions at 10-250 Gb/s have already been reported [28, 35]. It is expected that future devices could increase considerably this speed [16].

Acknowledgements

This research was supported by the Telecommunications Research Institute of Ontario and the Natural Sciences and Engineering Research Council of Canada.

References

1. Akl, S. G. On the power of concurrent memory access. *Proceedings of the International Conference on Computing and Information ICCI'89*. 1989, 49-55.
2. Akl, S. G. *Parallel Computation : Models and Methods*, Prentice Hall, 1997.
3. Aho, A.V., Hopcraft, J.E., Ullman, J.D., *The design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Mass., 1974.
4. Anderson, R.J., Miller, G.D., Optical Communication for Pointer Based Algorithms, Technical Report CRI-88-14, Computer Science Department, University of Southern California, 1988.
5. Ben-Asher, Y., Peleg, D., Ramaswami, R., Schuster, A. The power of reconfiguration, *Journal of Parallel and Distributed Computing*, vol. 13, 1991, 139-153.
6. Beame, P., and Hastad, J. Optimal bounds for decision problems on the CRCW PRAM. *Journal of the ACM*. **36**, 1989, 643-670.
7. Ball, G.A., Glenn, W.H., Morey, W.W., Programmable fiber optic delay line, *IEEE Photonics Technology Letters*, **6**, no. 4, 1994, 741-743.
8. Bilardi, G., Preparata, F.P., Horizons of parallel computing, *Journal of Parallel and Distributed Computing*, **27**, 1995, 172-182.
9. Chiarulli, D. M., Levitan, S. P., Melhem, R.G. Optical bus control for distributed multiprocessors, *Journal of Parallel and Distributed Computing*, **10**, 1990, 45-54.
10. Chiarulli, D. M., Melhem, R.G., and Levitan, S. P. Using coincident optical pulses for parallel memory addressing. *The Computer Journal*. Dec. 1987.
11. Guo, Z., Melhem, R. M., Hall, R. W., Chiarulli, D. M., and Levitan, S. P. Pipelined communications in optically interconnected arrays. *Journal of Parallel and Distributed Computing*. **12**, 1991, 269-282.
12. Guo, Z. Sorting on array processors with pipelined buses. *Proceedings 1992 International Conference on Parallel Processing*. 1992, III 289-292.
13. Guo, Z. Optically Interconnected Processor Arrays with Switching Capability. *Journal of Parallel and Distributed Computing*. **23**, 1994, 314-329.
14. Goldberg, L., Jerrum, M., Leighton, T., Rao, S., A Doubly-Logarithmic Communication Algorithm for the Completely Connected Optical Communication Parallel Computer, *Proceedings of the Symposium on Parallel Algorithms and Architectures*, 1993.
15. Guizani, M., Performance evaluation of optical systems for multistage interconnection networks, *Proceedings First International Workshop on Massively Parallel Processing Using Optical Interconnections*, 1994, 213-223.
16. Heuring, V. P., Jordan, H. F., and Pratt, J. P. Bit-serial architecture for optical computing. *Applied Optics*. **31**(17), 1992, 3213-3224.
17. Hamdi, M., Pan, Y., Efficient parallel algorithms on optically interconnected arrays of processors, *IEE Proceedings - Computer Digital Techniques*, **142**, no. 2, 1995, 87-92.
18. Fich, F., Ragde, P., Wigderson, A., Relations between concurrent-write models of parallel computations, *SIAM Journal of Computing*, **17**, 1988, 606-627.
19. Levitan, S. P., Chiarulli, D. M., and Melhem, R. G. Coincident pulse technique for multiprocessor interconnection structures. *Applied Optics*. **29**(14), 1990, 2024-2033.
20. Li, H., and Stout, Q. F. (eds). *Reconfigurable Massively Parallel Computers*. Prentice Hall, 1991.
21. Melhem, R. G., Chiarulli, D. M., and Levitan S. P. Space multiplexing of waveguides in

optically interconnected multiprocessor systems. *Computer Journal.* **32**(4), 1989, 362-369.
22. Marberg, J.M., Gafni, E., Sorting in constant number of row and column phases on a mesh, *Algorithmica*, 1988, 561-572.
23. Pan, Y. Order statistics on optically interconnected multiprocessor systems, *Proceedings First International Workshop on Massively Parallel Processing using Optical Interconnections*, 1994, 162-169.
24. Pavel, S., Akl, S. G., On the power of arrays with reconfigurable optical buses, *Proceedings of the International Conference on Parallel and Distributed Processing and Applications (PDPTA'96)*, August 9-11, 1996, 1443-1454.
25. Pavel, S., Akl, S. G., Matrix Operations using Arrays with Reconfigurable Optical Buses, *Journal of Parallel Algorithms and Applications*, **8**, 1996, 223-242.
26. Pavel, S., Akl, S. G., Area-Time Tradeoffs in Arrays with Optical Pipelined Buses, *Journal of Applied Optics*, **35**(11), 1996, 1827-1835.
27. Prucnal, P., Blumenthal, D., Perrier, P. Self routing photonic switching demonstration with optical control. *Optical Engineering*, **26**(5), 1987, 473-477.
28. Prucnal, P.R., Glesk, I., Sokoloff, J.P., Demonstration of all-optical self-clocked demultiplexing of TDM data at 250 Gb/s, *Proceedings First International Workshop on Massively Parallel Processing Using Optical Interconnections*, 1994, 106-117.
29. Qiao, C, and Melhem, R. G. Time-division communications in multiprocessor arrays. *IEEE Transactions on Computers.* **42**(5), 1993, 577-590.
30. Qiao, C, Melhem, R. G., Chiarulli, D.M., Levitan S.P. Optical multicasting in linear arrays, *International Journal Optical Computing*, **2**, 1991, 31-48.
31. Qiao, C. Efficient matrix operations in a reconfigurable array with spanning optical buses. To appear in *Parallel Processing Letters.*
32. Rajasekaran, S., Sahni, S., Sorting, Selection and Routing on the Array with Reconfigurable Optical Buses, Manuscript, 1995.
33. Rajasekaran, S., Sahni, S., Deterministic routing on the array with reconfigurable optical buses, Manuscript, 1995.
34. Sauer, J., A multi-Gb/s optical interconnect, *SPIE Proceedings, Digital Optical Computing II*, **1215**, 1990, 198-207.
35. Saito, T., Yano, Y., Henmi, N., Optical TDM 20 GB/s-105Km transmission employing newly proposed optical PLL timing extraction, *IEEE Photonics Technology Letters*, **6**, no. 4, 1994, 555-557.
36. Valiant, L.G., General Purpose Parallel Architectures, in *Handbook of Theoretical Computer Science*, J. van Leeuwen, ed., vol. A, North Holland, 1990.
37. Valiant, L.G., A bridging model for parallel computation, *Communication of the ACM*, **33**, 1990, 103-111.