Technical Report No. 95-374

On the Power of Arrays with Reconfigurable Optical Buses*

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

February 1st, 1995

Abstract

This paper examines some computational aspects of different arrays enhanced with optical pipelined buses. The array processors with optical pipelined buses (APPB) are shown to be extremely flexible, as demonstrated by their ability to efficiently simulate different variants of PRAMs and bounded degree networks. A model of computation is introduced, the *array with reconfigurable optical buses* (AROB), which combines some of the advantages and characteristics of the classical reconfigurable networks (RN) and the APPB. A number of applications of the APPB and AROB are presented, and their power is investigated. It is shown that beside AROB's capability of simulating classical reconfigurable networks, the enhanced communication mechanisms allow for an important system reduction when compared with the classical RNs.

Keywords: optical interconnections, pipelined optical buses, reconfigurable networks, bounded degree networks, PRAM models.

1 Introduction

Interprocessor communication networks are often the main bottlenecks in parallel machines. There are a number of fundamental constraints which bound bus interconnections in electronic systems in general: limited bandwidth, capacitive loading, and cross-talk caused by mutual

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

inductance. Another limitation 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 alternative solutions to these problems. Unlike the signal propagation in electronic buses, which is bidirectional, optical channels are inherently directional and have predictable delay per unit length. As it is shown in [21, 12], this allows a pipeline of signals to be created by the synchronized directional coupling of each signal at specific locations along the channel. The possibility in optics to pipeline the transmission of signals through a channel provides an alternative to exclusive bus access. 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 [21, 12, 27, 11, 29].

In this paper we analyze the possibility of using the arrays with optical pipelined buses in order to simulate different PRAM models and bounded degree networks. Our results complement the results previously obtained and extend the algorithmic study of these structures, making it possible to better understand the implications of using optical interconnections for massively parallel processing.

In order to overcome the inefficiency of long distance communications, the use of broadcasting bus systems has been proposed. When the configuration of such a bus system can be dynamically modified, the network is referred to as *reconfigurable*. A reconfigurable network is able to obtain a precise topology that mirrors the connectivity required by an algorithm through the use of data-dependent switch settings, [18]. We introduce in this paper a model which incorporates some of the advantages and characteristics of two existing models, namely the classical reconfigurable networks and the arrays with optical pipelined buses. The new array with reconfigurable optical buses, AROB, complements the communication and computation capabilities of the arrays with optical pipelined buses and switches given in [27, 29, 11], by using slightly different reconfiguration rules and functional characteristics. It is shown that beside the capability of the AROB to simulate classical reconfigurable networks, the enhanced communication mechanisms allow for an important system reduction when compared with the classical RNs.

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.

In section 2 the basic concepts used in this paper are presented. In section 3 the AROB model is introduced. Simulations of PRAMs and bounded degree networks by the APPB, and of reconfigurable networks by the AROB are described in section 4. Some applications are presented and the power of these models is investigated in section 5. Section 6 presents our conclusions and some open problems.

2 Basic Concepts

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

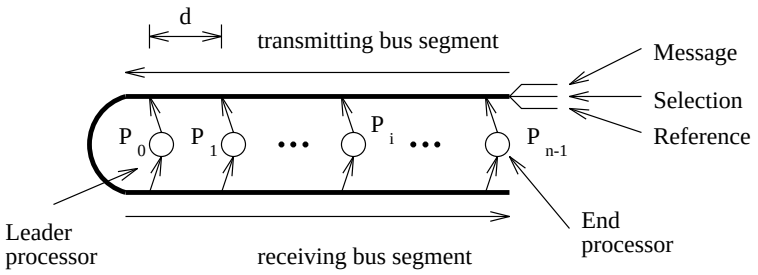

Figure 1: The linear array with pipelined optical buses APPB.

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*) 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.

Two conditions appear to be 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, is larger than or equal to bwc_g, 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 initiation of a consecutive transmission on the bus is possible only after a bus cycle ends. The *end-to-end propagation time* is $\sigma_b = 2n\tau$ seconds, where n is the number of processors in the linear array and τ, is the time taken for a message to traverse the optical distance d. This system is called linear *Array Processors with Pipelined Buses* (APPB), and its principles have been introduced in [21, 12].

Several approaches can be used to route messages in a linear APPB structure from one processor to another. In [21, 12] a time waiting function is introduced. In [9, 16, 28] a coincident pulse technique is described. Various message routing patterns can be realized using these techniques, most of which are suitable for both SIMD and MIMD implementations. Next we describe a version of the time-division multiplexing scheme, [27], which is also used to implement a part of the synchronous communications needed by the AROB. We consider two cases: (1) message receiving, when the destination processor knows the index of the sender and (2) message sending, when the sender knows the destination address.

Message receiving: To specify the time at which a processor should receive a message, a control function $wait(i, j)$ is introduced in [21, 12]. Function $wait(i, j)$ is defined as the time that processor i should wait, relative to the beginning of the bus cycle, before reading the

message sent on the bus from some other source processor j. Thus, $wait(i,j) = (i-j)\tau$. This technique assumes that the receiver knows the identity of the sender. In this communication scheme all processors start the transmitting phase at the same moment, that is, the beginning of a bus cycle. From the receiver's point of view, taking into consideration the receiving time for each transmitted message, each source processor has a fixed, exclusive, transmitting time slot. In [27], due to the one-to-one mapping between a source processor and the corresponding time slot this mechanism is called time-division source-oriented multiplexing (TDSM). Multiple processors can read the same message in the same bus cycle.

Message sending: Each message is written on the bus during the receiving time slot associated with its destination processor. Denote by s_i the receiving time slot associated to processor P_i, for all i, $0 \leq i \leq n-1$. The message communication process begins by making slot s_{n-1} available to P_{n-1}, on the transmitting segment of the bus. After τ time, slot s_{n-1} advances through the waveguide, reaching P_{n-2}, and slot s_{n-2} becomes available to P_{n-1}. This process is repeated, making all slots available to all processors. The train of time slots propagates, through the folded segment into the receiving segment, until s_{n-1} reaches the directional coupler of P_{n-1}. In this moment all processors read their messages on the receiving segment of the bus. Since there is a one-to-one mapping between a destination processor and a time slot, a technique similar to this is called time-division destination-oriented multiplexing (TDDM), [27]. An error can occur if several sources want to send messages to the same destination.

The address information can also be encoded using the *coincident pulse technique* presented in [9, 16, 28, 27], see Fig. 2. The *unit delay* is defined to be the spatial length of a single optical pulse, i.e. $w \times c_b$, [27]. Initially the processors are connected to the three waveguides of a bus such that between any two processors the same length of fiber is used in all three waveguides. Thus, the propagation delays are the same for all three waveguides, in both transmitting and receiving segments. It is also assumed that the bus interfaces can introduce delays between any two processors on the receiving (lower) segment of the bus. For the moment we consider that unit delays are added only to the reference and message waveguides. One such delay is represented by a loop in Fig 2.a. As a consequence, the propagation delays on the receiving segments of the select and reference waveguides are no longer the same.

The coincident pulse technique can be used when the source processor knows the address (index) of the destination processor. The address of a destination processor is unary encoded by the source processor in an *address* frame, which is composed of the *selection* and *reference* pair of frames, Fig. 2.b. This encoding is performed before the actual bus cycle. The time difference between the two pulses in the address frame, in units of delay, is selected to be equal to the absolute difference between the addresses (indices) of the source and destination processors. During the bus cycle, the source processor sends the address frame on the transmitting segment. Since there are no delays on the transmitting segment, the address frame arrives unmodified at P_0 on the receiving segment. After these two pulses propagate through their corresponding waveguides on the receiving segment, a coincidence of the two occurs at the destination processor, Fig 2.c. A *message* frame is sent on the message waveguide synchronously with the reference pulse. A simple hardware mechanism can be used by the destination processor in order to detect the coincidence of a reference and select pulses and to read, in this case, the message frame on the message waveguide. We denote by t_{ref} the time when processor

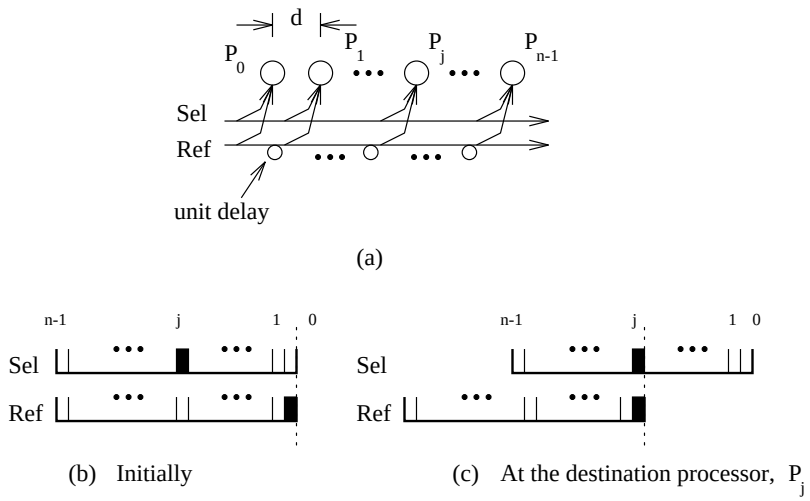

Figure 2: Coincident pulse technique. (a) Processor connection; the message waveguide and the transmitting segment are omitted. (b) Address frame (c) Pulse coincidence at the destination.

P_i transmits its reference pulse and by $t_{sel}(j)$ the time when it transmits a select pulse. These two pulses will coincide at processor P_j if and only if $t_{sel}(j) = t_{ref} + j$, $0 \leq i, j < n$.

A selection frame contains n pulse slots, each corresponding to a destination processor. The presence of a pulse in one of these slots specifies the address of a destination processor. Thus, multiple processors can be addressed by a single address frame if multiple pulses are specified in the selection frame. For example, a processor can use the same addressing mechanism in order to broadcast a value to all other processors [25]. In this case if all the slots in the selection frame contain optical pulses then a coincidence between a selection pulse and the reference pulse will occur at each processor connected to the bus. Many other communication patterns are described in [9, 16, 28] using the same coincident pulse technique. The decoding of the destination address is done through the detection of the coincidence of two pulses at the destination processor. The coincident pulse mechanism seems to avoid the traditional bottleneck which characterize most of the on-line address decoding techniques [27].

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 operation includes message generation and detection, synchronization, buffering, etc. [12]. 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, of message processing time to communication over distance d time is appreciated to be on the order of 10 to 1000, see [12]. In order for the message processing time to be in 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 $O(\sigma_b)$.

In order to allow multiple source processors to initiate communications in parallel, the overlapping between the *packets* sent by different source processors during the same bus cy-

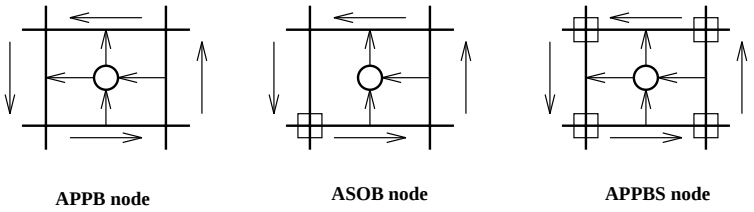

Figure 3: 2D arrays with optical pipelined buses and different switching capabilities.

cle, should be avoided. A *packet* is represented by the message frame and the address frame (selection and reference pulse frames) sent by a processor on the bus. Thus, the condition $d > \max\{b, n\}wc_g$ must be satisfied. For big values of n the length of the address frame (n) could be much larger than the length of the message frame (b), resulting in very long buses which are inefficient.

There are also other factors that limit the size of an optical bus. All passive devices, as the directional couplers or optical fibers, exhibit loss. In the same time each directional coupler, used by a processor to read a message from the waveguide, taps a percentage of the energy carried by the signal. One solution for these problems is to provide some means of amplitude restoration. The use of an active switch instead of the directional coupler could solve this problem, [13], but it has the main disadvantage that introduces more delay. Otherwise, the number of the devices connected to a bus must be restricted to an admissible value, i.e., which does not introduce errors. This second solution reduces also the value of the end-to-end propagation delay. 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 from $O(n)$ to $O(\sqrt{n})$.

2.2 2D array processors with optical pipelined buses and switches

The two-dimensional arrays presented in [27, 29, 11] are versions of the same structure, called *Array Processors with (Optical) Pipelined Buses* (2D APPB) [12], enhanced with different switching capabilities. The system used in each row and each column of a 2D APPB is the *linear* APPB [21, 12]. The connection of a 2D APPB processor is depicted in Fig. 3.

A node of the *Array with Spanning Optical Buses* (ASOB) presented in [27, 29] is also shown in Fig. 3. This structure is based on the utilization of $2n$ linear APPBs, each used to interconnect the processors in a row or a column. One segment of each row optical bus is connected through a switch to one segment of each column optical bus. An extended variant of this structure, the *Array Processors with (optical) Pipelined Buses using Switches* (APPBS), is presented in [11]. This architecture is similar to that of the basic two-dimensional APPB and also to that of ASOB. The difference lies in the use of optical switches at the intersections between all bus segments of the column and row buses, see Fig. 3. Also, different switching rules are used.

In the next subsection we use the basic concepts presented for the APPB in order to describe another two-dimensional array which uses optical pipelined buses for communication, the AROB.

3 Two-dimensional Array with Reconfigurable Optical Buses

The *Array with Reconfigurable Optical Buses*, AROB, we propose uses the basic architectural and functional structure of a classical reconfigurable network. 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 [21, 12] and also presented in the previous section.

3.1 Reconfigurable networks

A linear reconfigurable network consists of n processors and has a simple topology. The processors $P_0, P_1, \ldots, P_{n-1}$ are arranged in a row. Switches are used to connect processor P_i to processors P_{i-1} and P_{i+1} if $1 \leq i \leq n-2$. Thus, for a switch of a linear network there are only two local configurations: either connect or disconnect the edges corresponding to the two adjacent processors. Many reconfigurable two-dimensional network architectures have been proposed in the literature. These include the polymorphic torus [17], the reconfigurable mesh with buses (RMESH) [23], the processor array with a reconfigurable bus system (PARBUS) [32], and the reconfigurable network (RN) [6]. An $n \times n$ PARBUS [32] consists of a mesh of processors each of which is connected to its neighbors through links, as shown in Fig. 4.a. The up to four interprocessor links to which a processor has access can be connected internally in arbitrary subsets. This can be done by an internal switching system associated with each processor. Of particular interest here is a modified version of this model, namely the RN, which is proposed in [6]. In the RN model the processors and the bus links are exactly as for the PARBUS. However, there are only ten local configurations that may be taken by a switch of the mesh. These configurations are depicted in Fig. 4.b.

A common basic assumption regarding any reconfigurable model is that the transmission on any bus obtained through reconfiguration takes constant time. This is based on the experimental results obtained with the YUPPIE architecture [22] for a systems using a few hundreds of processors, see also [18]. The above assumption ignores the fact that up to $O(n^2)$ processors can be connected to a bus and also ignores the fact that all electrical signal propagation problems described in the introduction worsen with the increase of the number of processors. In order to alleviate some of these problems, and also to make the constant time delay assumption more realistic, the replacement of electrical buses and switches with optical ones has been proposed in [6]. This optical reconfigurable architecture does not allow *message pipelining*.

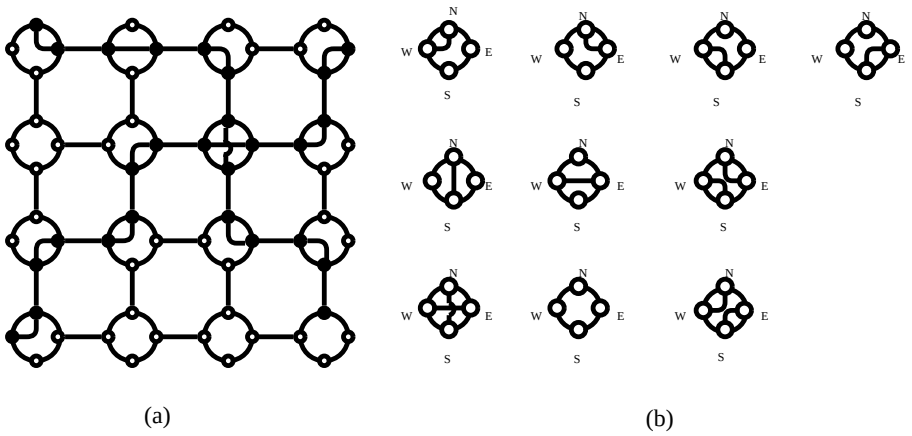

Figure 4: (a) Two-dimensional reconfigurable network; (b) Switch configurations.

3.2 The Two-dimensional AROB

Consider n^2 processors connected in a two-dimensional reconfigurable structure as depicted in Fig. 4.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. 4.b. The interconnection network is assumed to use optical waveguides and optical switches for communication and reconfiguration, respectively. We call this structure *Array Processors with Reconfigurable Optical Buses*, AROB. We describe in this section some of the most important functional and architectural characteristics of the AROB. Specifically, we show that the local switch setting provides a mechanism for building arbitrary APPB-like linear structures on the AROB. As one of the main objectives is to exploit the great communication capabilities of these linear APPBs, a number of basic requirements which must be satisfied in order to be able to implement time division and/or coincident pulse techniques, are identified.

A bus link between the ports of two neighboring processors is composed of two segments, the signal propagation direction on each of the two bus segments being indicated with arrows in Fig. 5.a. A processor sees these segments as *transmitting* and *receiving* segment, respectively. Note that a transmitting segment of a processor is in the same time a receiving segment for its neighbor. Each transmitting or receiving segment contains all three waveguides, i.e., the message, selection and reference waveguides. When required, delays can be introduced in the path of any of these waveguide.

Linear bus construction

If during a switch setting operation two ports of a processor must be connected, then the receiving segment from one port is connected internally to the transmitting segment of the other port and vice versa. Another possible case is when two or all four ports of a processor remain unconnected.

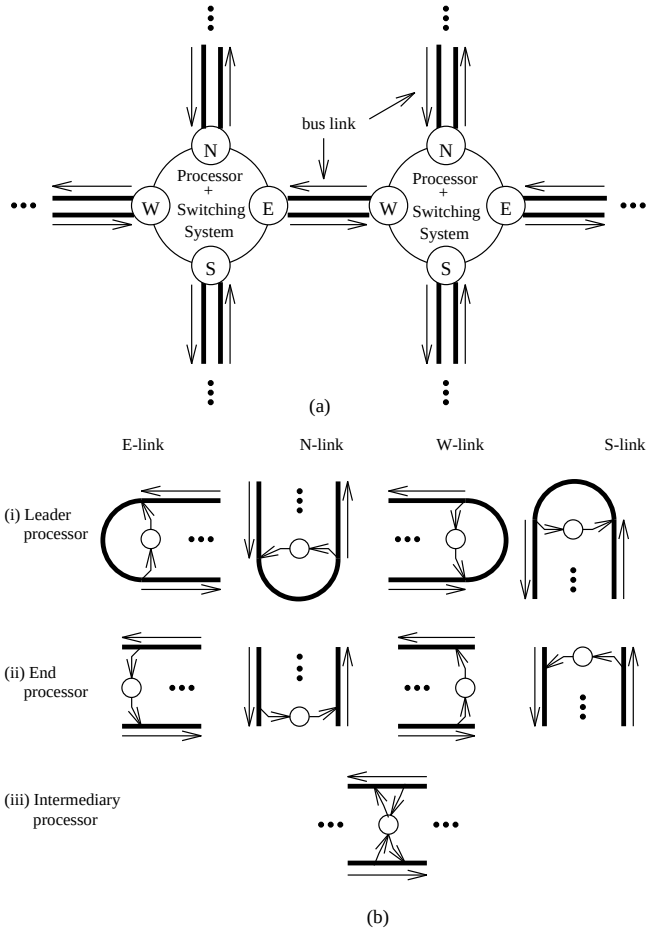

Figure 5: (a) The processor connection in a 2D AROB. (b) Possible processor connection to an arbitrary bus.

When pipelined communications are sought, an algorithm to be implemented on the AROB is required to specify a rule in order to uniquely identify the *leader* processor of each linear array obtained by reconfiguration (similar to P_0 in Fig. 1). It is assumed also that the designated leader is able to connect a folded optical bus between the transmitting and receiving segments of the port through which it connects to that bus.

Thus, multiple arbitrary linear structures, similar to the linear APPB, can be constructed through reconfiguration. A processor can be connected to up to four such buses at a time. The algorithms to be simulated on the AROB are not necessarily required to specify to each processor the length of the linear buses to which it is connected or its indices relative to these buses. Note that these values are needed in order to be able to implement the time division or coincident pulse techniques. If they are not specified, the technique described below can be used in order to determine the relative position of each processor in each linear bus to which it is connected and, implicitly, the length of each bus.

There are essentially three different ways of connecting a processor to a linear structure obtained through reconfiguration, Fig. 5.b. As the reconfiguration process is the result of some local decision, after the switch setting operation each processor knows if it is either an intermediary, or a terminal processor in the linear bus obtained. If two ports of a processor are connected internally then that processor is described as intermediary for the corresponding bus. Any port which is not connected internally to another port designates the associated processor as terminal. When a processor is determined (or specified) as the leader of a bus, its connection is similar to one of those depicted in Fig. 5.b (i). Any terminal processor which is not a leader is an end processor and its connection is one of the four depicted in Fig. 5.b (ii). At the time of reconfiguration there is no way of knowing by an intermediary processor what is its position in the linear bus or, which of the two bus segments is the transmitting and which is the receiving one in the new linear APPB-like bus. Another bus cycle is required in order to determine the receiving and, implicitly, the transmitting segments of the linear bus. The processor connection given in Fig. 5.b (iii) allows the processor to monitor both bus segments for signals coming from either of the two directions. The leader processor sends a pair of pulses on the transmitting segments of the selection and reference waveguide, respectively. Note that as the propagation directions on the bus segments are predetermined, the leader knows their function designation. The optical pulses propagate through the folded segment and then into the receiving segment of the bus. As they travel synchronously their coincidence is detected by each processor connected to the bus. The particular segment of the bus on which these pulses are detected is set to be the receiving bus segment.

Arbitrary buses which connect $O(n^2)$ can be obtained through reconfiguration. Unless otherwise state, in order to avoid the problems created by long buses we restrict the use of the AROB to applications which require buses of $O(n)$ length. Also, it is assumed that each processor in an AROB has $O(1)$ memory. The limited memory restricts to $O(1)$ the number of different messages sent and/or received in a bus cycle by a processor.

Index determination

We describe next a technique which can be used in order to determine the index of each processor connected to a linear bus obtained in an 2D AROB by reconfiguration. Consider the linear array depicted in Fig. 6.a. The leader processor is the only one in this structure which knows its index (0). Note that the transmitting and folded bus segments are not shown in Fig. 6.a, and that the message waveguide is entirely omitted. Each processor, with the exception of P_0, introduces a unit delay on the receiving segment of the selection waveguide. This is followed by a bus cycle at the beginning of which P_0 sends a pair of pulses on the selection and the reference waveguides. The pulses arrive at P_0 on the receiving segment at moment t_0. From one processor to the next the selection waveguide has added unit-delay loops. The difference between the arrival times of the selection and reference pulses is incremented with one time unit at each new processor. The selection pulse arrives at the jth processor with j time-units delay relative to the reference pulse. The reference pulse is used to trigger at each processor the load of the selection frame. Thus, the relative delay, between the two pulses gives the actual index of each processor in the linear structure.

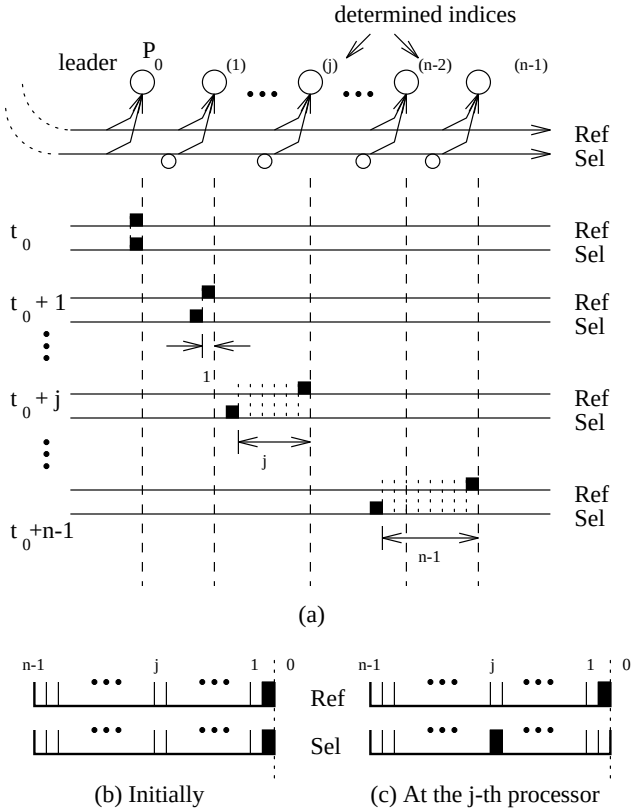

Figure 6: Index determination.

This is actually the reversed process of the coincident pulse addressing technique presented in the previous section. It starts with two synchronized pulses, Fig. 6.b, and after a number of delays the unary-encoded information (the index) is obtained at each processor, Fig. 6.c. In this case it is no longer required that the length of a packet to be increased to $\max\{b, n\}$ as is the case with the coincident pulse technique. No more than one pulse frame is sent on a waveguide during a bus cycle, and thus the possibility of collision is eliminated.

Comparison with other arrays with optical pipelined buses

The ASOB [27, 29] and APPBS [11] structures are basically array processors with pipelined buses, enhanced with different optical switches at the intersections of row and column buses. Compared with the 2D APPB, these switching capabilities allow improved communications between the rows and columns. The use of switches provides all-optical paths between any pair of processors, reduces the number of message transfer steps and reduces or even eliminates the buffers used in the basic 2D APPB. Our approach in combining reconfiguration and optical message pipelining is slightly different. The AROB architecture has two major objectives: (1) to be able to simulate efficiently the reconfiguration mechanism of a reconfigurable network (RN), and (2) to add to the reconfigurable network the great optical communication capabilities of the APPB-like structure.

(1) The simulations of RNs by AROBs are investigated in section 4.2 where it is shown that an AROB with n^2 processors can efficiently simulate any n^2-processor RN. One advantage of the reconfigurable networks is the possibility to build arbitrary buses which allow direct connections between different points in the network. If this would be the sole purpose of the AROB, our model would not bring anything new because direct connections between different nodes can also be obtained, under certain conditions, using the ASOB or the APPBS. The power of the reconfigurable networks is also given by the possibility of using the bus reconfiguration as a *computation tool*. In order to illustrate this claim we give as example the summation algorithm presented in [15]. It is shown in [15] that given N k-bit numbers, $1 \leq k \leq N$, these numbers can be added in $O(1)$ time on an $N \times Nk$ reconfigurable network. Besides the usual binary representation of numbers within a processors, the data are also represented using the input/output ports of a group of processors in the reconfigurable mesh. The 1UN and 2UN representations use n processors to represent an integer i in $[0, n-1]$. The designated port of processor k, $0 \leq k \leq i$ carries a 1-signal and the designated ports of the rest of the processors carry a 0-signal, [15]. See the left-most column of the array in Fig. 7. For the 2UN representation of an integer i, the designated ports of some subset of i processors carry a 1-signal and the designated ports of the rest of the processors carry a 0-signal. See the first three processors of the top-most row in Fig. 7. Note that the 2UN representation of an integer is not unique. In Fig. 7 is shown one block of the mesh which is used to compute $1\text{UN}(C_{j+1}) = (2\text{UN}(S_j) + 1\text{UN}(C_j))$ div 2, where S and C represent sum and carry values, respectively. The $2\text{UN}(S_j) + 1\text{UN}(C_j)$ and div 2 operations are performed by the appropriate bus setting and the result is obtained in the right-most column after the 1-signals propagate through the buses. See [15] for details.

(2) The APPB communication capabilities can be exploited by setting the AROB's switches

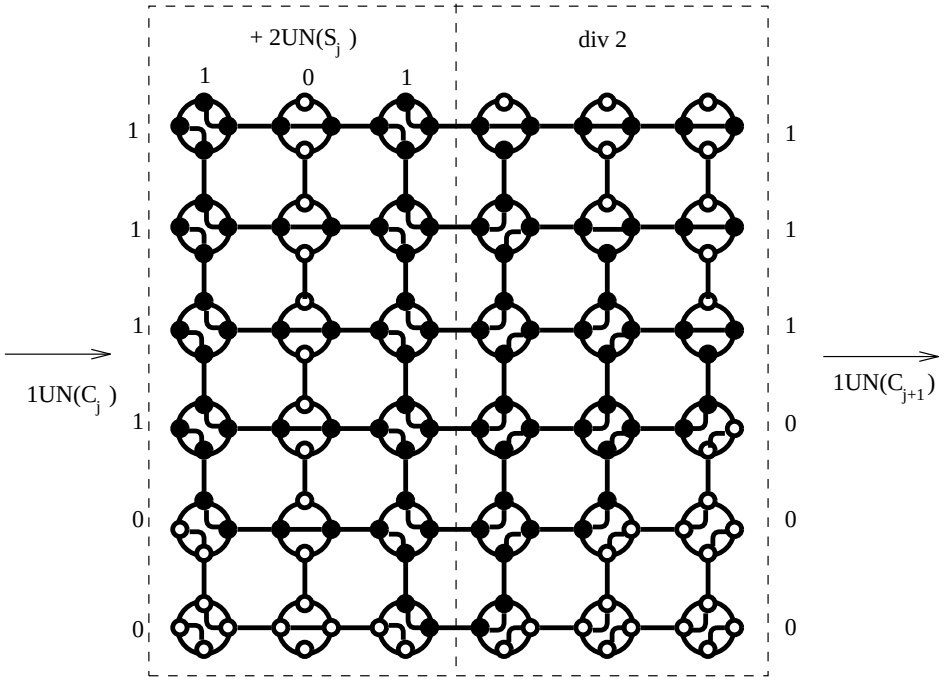

Figure 7: Bus configuration for the computation of $1\mathrm{UN}(C_{j+1}) = (2\mathrm{UN}(S_j) + 1\mathrm{UN}(C_j))$ div 2 for $C_j = 3$ and $S_j = 2$.

such that multiple processors of the AROB are connected in "contiguous" linear and/or two-dimensional APPB-like structures. For example, this type of communications is used in order to obtain an efficient 2D compaction algorithm, see claim 8.

The ASOB and the APPBS systems cannot be easily and directly compared with the AROB because they use different switching techniques. Some of the communication patterns implementable in the APPBS cannot be simulated on an AROB with exactly the same time complexity. For example, the AROB model, unlike APPBS, does not allow switch setting operations to take place in the same time with the message transfers during a bus cycle. Minor modifications in the hardware and functional specification of the switching system of an AROB are required in order to implement the communication patterns of the APPBS, with exactly the same time complexity. In the same time, the switching system used in APPBS allows the implementation of only a subset of the switch state configurations specific to a RN system. Again, only minor modification are needed in order to transform the APPBS into an AROB.

3.3 Time complexity issues

Evaluating the time complexity of algorithms simulated on structures which use some kind of global, fixed or reconfigurable, broadcasting buses remains a controversial issue. Initially the propagation delay on a global (electrical) bus was expressed as a function of its length, i.e., the number of processors connected to it [5]. It was noticed later that the propagation time on the bus can be considered constant for practical values of the number of processors [22]. This assumption is systematically made when reconfigurable networks are used. In reality the communication time increases with the number of processors in the system, regardless the technology employed. This is true even if other communication techniques, like the free-space optical beams, are used.

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, [10]. This method has the advantage that it abstracts from the details introduced by the technology dependent parameters, as σ_b, for example. Obviously this is true only if the same assumptions regarding the bus cycle lengths are made. The assumption made in [21] is that the bus cycle length in a linear APPB is compatible with the computation speed in the nodes. In [12, 11] and [25] the time complexity of different algorithms simulated on arrays with optical pipelined buses is also expressed in terms of the number of steps and it is shown that these results compare favorable with those obtained on different classical networks.

We adopt in this paper the same method of specifying the time complexity in number of steps. However, one must be aware that for a more accurate comparison of our results with those obtained on other architectures, as the RN or the mesh with global broadcasting buses for example, would imply the explicit use of the bus cycle length parameter, $O(\sigma_b)$. Obviously, the communication delays encountered in the actual implementations of the other architectures should also be accounted for.

4 PRAMs, BDNs and RNs simulations

In this section we investigate the power of the linear APPB in relation to some of the PRAM models and bounded degree networks (BDN). We give two algorithms to simulate the Common- and Priority-CRCW PRAM, respectively, on a (slightly modified) linear APPB for the case when each processor has constant-size memory, i.e., the total memory in the system is $m = \Theta(n)$. These simulations are further extended, using a two-dimensional APPB, to the general case $n = o(m)$. This second technique allows us to also give a simulation algorithm for the Combining-CRCW PRAM model. It is shown further that the 2D APPB structure can efficiently simulate any bounded degree network which has the same number of nodes. Finally, the simulation by the AROB of the RN, is presented. The time complexities of these simulations are also analyzed.

4.1 PRAM simulations using the linear APPB

An (n, m)-PRAM consists of n processors and m memory locations, where each processor is a random-access machine, see for example [2]. All processors communicate via the shared memory. Each processor may read or write a memory location or perform any logical or arithmetical operation on some date stored in its local memory. There are different types of PRAMs depending on whether or not concurrent access to the same memory location is allowed. In this paper we consider the model variants which allow concurrent read (CR) operations. For the write operation, the simulation of both exclusive (EW) and concurrent (CW) cases are investigated. 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 are:

- Priority-CRCW PRAM: the processors are assigned fixed priorities, and only the one with the highest priority is allowed to write in a memory location in case of conflict;
- Common-CRCW PRAM: in case of conflict, the processors wishing to write into some memory location are allowed to do so only if they are attempting to write the same value, otherwise the value in that memory location remains unchanged;
- Combining-CRCW PRAM: the value written in a memory location is a linear combination, using some associative and commutative operation, of all values which were concurrently written.

First we consider the linear APPB structure, Fig. 1, with n processors, each with $\Theta(1)$ memory. Part of this local memory is used to implement the PRAM's global memory. We want to simulate any operation of an n-processor PRAM model having $m = \Theta(n)$ memory. Let both the linear APPB and the PRAM use processors of the same type. Thus, any internal arithmetic and/or logic operation of a PRAM processor is executed in the same amount of time by a APPB processor. It remains to show how to simulate the PRAM memory read/write operations and to give their time complexity.

If in the *message receiving* and *message sending* procedures, described for the linear APPB, the messages are replaced by PRAM memory values, we already have the CREW PRAM simulations algorithm. Indeed, the message receiving cycle can be used to implement a PRAM memory concurrent read operation and the message sending cycle to implement a PRAM memory exclusive write operation. In order to avoid contention during the write operation, either the algorithm (program) ensures that no two processors write in the same location in the same bus cycle or, a mechanism to detect possible errors is used. An example of such an mechanism is presented bellow for the Common CRCW PRAM which can also be used for the CREW PRAM. Thus, an n-processors linear APPB can simulate any n-processor and $\Theta(n)$-memory CREW PRAM operation in a constant number of bus cycles and $O(\sigma_b)$ time. For the simulation of the Common- and Priority-CRCW PRAM models, we rely on two algorithms for the following problems.

Subset binary OR problem: Given a linear APPB with n processors each having a one-bit value v_i, for $0 \leq i \leq n-1$, it is required to compute the binary OR of the values stored by an arbitrary, specified, subset of processors S. The result should be stored in memory location m_j which is associated with some processor P_j. The index j of this processor is known to all processors in S. Because in a linear APPB during a write cycle all processors have writing access to each message frame, a single memory write cycle suffices in order to compute the subset binary OR value. Indeed, during the memory write cycle each processor $P_i \in S$ writes its binary value on the transmitting segment of the bus when the jth message slot reaches that processor, that is, after $(n-i+j)\tau$ time from the beginning of the bus cycle. The directional coupler used to connect a processor to a waveguide acts as an optical equivalent of a wired OR gate. The result, which obviously is the binary OR of all these values, is read by P_j on the receiving segment of the bus after $(n+i+j+2)\tau$ time from the beginning of the cycle. □

The coupler can be replaced by a true OR gate made from an active switch when interference between coherent input signals is a concern, [13]. A true OR circuit provides in the same time synchronization and signal amplitude restoration, [13], but has the disadvantage that introduces its switching time as delay on the optical path.

Subset binary prefix OR problem: A binary prefix OR operation over one-bit values, v_{j_k}, stored by an arbitrary subset of processors S of a linear APPB requires the computation of $b_i = v_{j_0} \vee v_{j_1} \vee \ldots \vee v_{j_h}$, with $j_k < i$, for all k, $0 \leq k \leq h$, $h < i$, for all processors $P_i \in S$. The indices associated with each processor (and value) are given by the relative position in the linear APPB. Thus, the subset binary prefix OR problem requires to compute for each processor $P_i \in S$ the OR function of the one-bit values stored by all the processors in S which have indices smaller than i. The subset S has associated a work memory location m_x, for some $x \in [0, n-1]$ which is initialized to zero. The index x is specified by the algorithm and is known by all processors in S. The subset binary prefix OR problem can be solved by a linear APPB where the processors are connected as in Fig. 8. At the beginning of the bus cycle P_x sends m_x on the transmitting segment of the bus. All other message slots remain unoccupied. When the message frame associated with m_x reaches some processor $P_{j_k} \in S$ on the receiving segment, P_{j_k} reads the value stored by m_x. The read operation is performed using the input directional coupler A after $(n+j_k)\tau$ seconds from the beginning of the cycle. When m_x reaches the output directional coupler B of the same processor, the v_{j_k} is written on the bus. The value

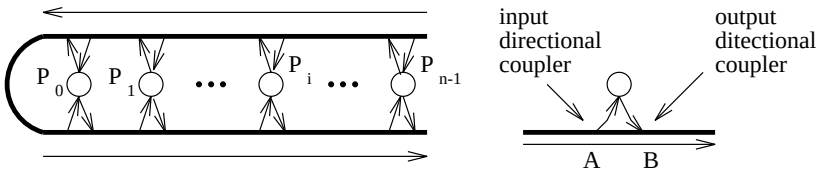

Figure 8: Processor connection for the BPS problem.

read by each processor $P_i \in S$ is the OR of the one-bit values, v_{j_k}, stored by the processors in S which have indices j_k smaller than i. A similar technique can be used if the order of indices is reversed, i.e. if the prefix OR of the values from the right of each processor are needed. In this case the transmitting segment is used similarly in a TDDM cycle and the prefix OR values are obtained in one bus cycle. □

Multiple binary OR and prefix OR values can be computed in the same bus cycle if their destination addresses are distinct and the subsets of processors involved are disjoint. These algorithms take a constant number of steps.

Different variants of the Common-CRCW PRAM model are mentioned in the literature, see [2] for example. The basic definition of the Common-CRCW PRAM does not require the machine to check if the values, written by a subset of processors in the same memory location, are indeed equal. Instead, the model requires this to be guaranteed by the algorithm. Thus, we could use the message send/receive procedures to implement the Common-PRAM memory read/write operations. Other variants require that the integrity of a value written in a memory location be checked and some decision be taken accordingly. For example, the COLLISION Common variant requires a "failure" label to be stored in the memory location in case the concurrent write does not succeed, i.e., the processors attempted to write different values in the same location. Next we provide a mechanism for checking the integrity of a value written in a memory location. This can further be used to implement different variants of the Common model.

Suppose that a subset of processors wish to write in the same memory location m_j, stored by P_j. A write bus cycle is used,. This is followed by a read bus cycle during which each processor of the above subset reads the result stored in the m_j memory location and compares it with the value it has written previously. If for some processor these two values differ, than an internal flag is set. Another bus cycle is used in order to compute the binary OR value of all these flags, the result being also stored by P_j. Each such flag could be used individually or combined with other similar flags in order to determine the decision to be taken by the particular variant of the Common-CRCW PRAM simulated. For example, in the COLLISION Common model all the processors which have detected an error, use one more bus cycle to write the same failure label in the corresponding memory location. Thus,

Claim 1 *An n-processors linear APPB can simulate any Common-CRCW $(n, \Theta(n))$-PRAM operation in a constant number of steps.* ■

To simulate a Priority-CRCW PRAM we want to implement a mechanism to ensure that only the processor with the highest priority, from the subset of processors wanting to write in the same memory location, can write in that memory location. The priority is given by the position on the bus and the processor with the highest priority is the *leader*, P_0, and that with the lowest priority is the *end* processor, P_{n-1}. The simulation of a Priority-CRCW PRAM memory write operation consists of three steps.

The first step is a multiple binary prefix OR cycle. One binary prefix OR computation is associated with each memory location m_j which is to be written later during this Priority concurrent write operation. Only those processors which want to write in a given memory location are supposed to take part in the binary prefix OR operation associated with that memory location. Initially they set an internal flag to 1. In the next bus cycle these flags are used in a binary prefix OR operation. During the (multiple) binary prefix OR operation each processor wanting to write into m_j, reads a 1 value if and only if there is another processor with higher priority which also wants to write into m_j. Then, each processor which has to write into m_j checks the value read in the previous step. If this is 1 then the processor inhibits itself. Otherwise, it is the processor with the highest priority wanting to write into m_j. In the third step, which is a writing cycle, the unique processor with the highest priority writes into the m_j memory location. This is done in parallel for all memory locations to be written.

Claim 2 *An n-processors linear APPB can simulate any Priority-CRCW $(n, \Theta(n))$-PRAM operation in a constant number of steps.* ■

The above results have mainly theoretical significance because it is assumed that $m = O(n)$. In practice it is expected that any realistic parallel machine to simulate the PRAM has a fixed number of processors, say n, each having associated a memory module of size $O(m/n)$, where m is the maximum memory required by any algorithm to be implemented on this machine. By "realistic parallel machine" we mean one with distributed memory and an interconnection network of fixed degree. Clearly, a 2D APPB architecture with n processors and m memory, $n = o(m)$, satisfies this description. The simulation techniques used when $m = \Theta(n)$ can no longer be used for $n = o(m)$. In order to do this, we use a number of intermediate results. We show that any machine which uses a bounded degree interconnection (electrical) network G for communications, can be simulated in a constant number of steps on a two-dimensional APPB with the same type and number of processors. An immediate implication of this is the possibility to simulate, in a constant number of steps, an n-processor butterfly network [19] on an n-processor APPB. Further, this allows us to use Ranade's result [30] and give a simulation algorithm for any variant of the CRCW (n, m)-PRAM, on a 2D APPB. This algorithm has a slowdown factor of only $O(\log\ n)$.

Bounded degree network simulation on arrays with optical pipelined buses

It is mentioned in [12] that any arbitrary one-to-one permutation can be implemented in a two-dimensional APPB in a three-phase routing approach. The first step of this routing is a preprocessing step which redistributes messages in each row (column) such that the messages going to the same row (column) will occupy different columns (rows). Then, the second and

third steps will route the messages to their destination rows (columns) and columns (rows), respectively. The entire routing takes 3 bus cycles. A somehow similar three-phase off-line permutation routing algorithm is given in [1] and is described also in [19] for different types of classical networks. These techniques use the (off-line) construction of a bipartite graph which is further partitioned into perfect matchings in order to determine the initial preprocessing permutation in each row (column). The same technique can be used for off-line permutations on higher-dimensional meshes and on networks like the butterfly [1], for example. It turns out that it can also be used to implement the simulation of any constant degree network on a two-dimensional APPB. For completeness, in what follows we give a short description of this technique, the details of implementation being similar to those in [1, 19], for example.

We show now how to simulate the communication steps which take place in any n-node network G with maximum degree k, on an n-processor two-dimensional array with optical pipelined buses. Assume that the ith node of G is initially mapped to the ith processor of the APPB, where for example, the ith processor of the APPB is (r, c) with $i = (r-1)\sqrt{n} + c$, and $1 \leq r, c \leq n$ and $1 \leq i \leq n$. Clearly, the neighboring nodes of G might not be mapped to neighboring processors of the APPB. Thus, we need to simulate the single step communications which take place in the G network between any two neighboring processors. For this, we construct a bipartite graph of G with $2n$ nodes and with the edges representing the neighbor-to-neighbor communications in G. It is shown in [19], for example, that we can label the edges of the bipartite graph with integers from $[1, k]$ so that no pair of edges with the same label are incident to the same node. This labeling can be used to identify up to k disjoint one-to-one permutations which describe the communications in the G network. Thus, by applying k permutation routings, we have simulated the communications in G. As the preprocessing is done off-line, the permutations take a constant number of steps on an array with optical pipelined buses.

Claim 3 *The communications which take place in any constant degree, n-node network, can be simulated on an $n^{1/2} \times n^{1/2}$ APPB in a constant number of steps.* ■

One important consequence is that an n-node two-dimensional array with optical pipelined buses is *universal*, in the sense that it can simulate, in a constant number of steps, the majority of the networks which have been proposed as possible practical solutions for the interconnection networks of parallel systems. It is well known that networks like the shuffle-exchange, for example, cannot be efficiently implemented using VLSI technology, that is, without having long wires for some of the links in the network. However, the use of arrays with optical pipelined buses offers an opportunity to increase the efficiency with which such networks can be simulated, yet with a reduction of the communication system complexity. Many of the emulation results presented in [12, 11] for different classical networks are better than this three steps technique. Our result is a proof of principle rather than a constructive method. For example, the simulation of an n-node butterfly-like network we can be done even more efficient than in the general case of constant degree networks. Suppose that we want to simulate a butterfly with $(l+1)$ levels and $n = (l+1)2^l$ nodes. A simple and efficiently layout of the array with optical pipelined buses to simulate the butterfly is the $(l+1) \times 2^l$ array. In this case the communication patterns are

straightforward, no off-line preprocessing being necessary.

It is shown in [30] that an n-processor butterfly-like network with $O(m)$ memory can simulate one step of a CRCW (n, m)-PRAM in $O(\log n)$ steps with high probability. Thus,

Claim 4 *One step of a CRCW (n, m)-PRAM can be simulated on an $n^{1/2} \times n^{1/2}$ array with optical pipelined buses in $O(\log n)$ steps with high probability.* ■

Note that the concurrent write operation can be any of the Priority, Common or Combining, and that there is no limit imposed on the number of memory locations, m.

4.2 RN simulation by the AROB

Consider an AROB with n^2 processors. We want to simulate a reconfigurable network (RN) using the AROB model. We assume that processors are identical and thus, any internal or switch setting operations take the same time on both models. The only operation we have to simulate is the broadcasting of a value by an arbitrary processor, on the bus to which that processor is connected. An example of processors connected to a bus obtained after the switch setting operation is presented in a simplified manner in Fig. 9. Note that in this case no leader processor needs to be specified.

As it is the case with the classical reconfigurable networks [6], an error should be detected and reported if two or more processors attempt to broadcast a value during the same bus cycle. The broadcasting of a value is done in three bus cycles. In the first bus cycle each processor which wants to broadcast sends a pair of selection and reference pulses on both segments of the bus. If a processor detects at least one pulse coincidence on either of the two bus segments, sets an internal flag to 1, otherwise the flag is set to 0. If there is a processor which wants to broadcast and in the same time has the internal flag set to 1, then a broadcasting error is detected. Note that there are at least two processors which have detected such an error, if any. A similar bus cycle is used in order to inform all processors connected to the bus that an error has occured. In this case the coincidence of at least two pairs of pulses are detected by each processor on the bus. If no error has occured then no pair of pulses is sent during this last cycle, and the next cycle is used by the unique broadcasting processor to send the data to all other processors using both segments of the bus.

The coincident pulse technique [9, 16, 25] (see the previous section) can be used for this broadcast, only that this requires packet frames of $\max\{n, b\}$ length. A better technique is to send the message frame, containing the datum to be broadcast, synchronously with a pair of reference and selection pulses. The detection of the coincidence of the two pulses triggers the load of the message frame by each processor. Thus,

Claim 5 *An AROB with n^2 processors can simulate any n^2-processor RN operation in a constant number of steps. If the propagation times on buses of the same length in the two systems are identical, than the simulation is time and cost optimal.* ■

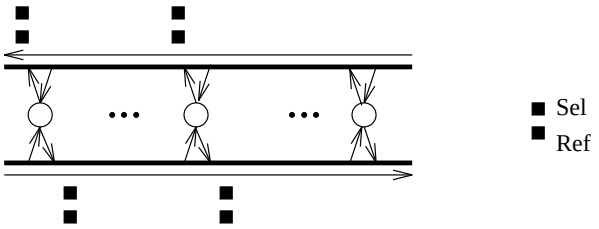

Figure 9: Processors in an arbitrary bus obtained through reconfiguration.

For this RN simulation using the AROB model, the algorithms are no longer required to specify the *leader* of each broadcasting bus. This requirement appears only when time division communications are needed. It is important to emphasize the differences between RN and AROB. By contrast with the classical RN, the AROB structure can use pipelining in order to allow multiple broadcasting of different messages on the same arbitrary bus, during the same bus cycle. The internal time-related operation and the coincident pulse techniques, represent other important enhancements over the classical RN. The relative powers of the AROB and RN are more formally presented below in claim 10.

5 Applications and Performances

5.1 Binary Prefix Sums on a linear APPB

Given n one-bit values, v_i, $0 \leq i \leq n-1$, the *binary prefix sums* (BPS) problem requires the computation of $ps_i = v_0 + v_2 + \ldots + v_{i-1}$, for all $0 \leq i \leq n-1$, where "+" is the addition operation. Consider the linear APPB structure with n processors, each storing one bit of the input sequence.

We show that this structure can compute the binary prefix sums of n one-bit values in a constant number of steps. The BPS is a simple adaptation of the *index computation* algorithm used by the processors connected to a bus obtained on an AROB after reconfiguration. The only difference is that initially only the processors whose one-bit values are 1 introduce a unit delay on the selection waveguide of the receiving segment of the bus, Fig. 6. At the beginning of the bus cycle P_0 sends a pair of pulses on the selection and reference waveguides. The time difference between the two pulses is incremented with one unit at each processor which has a 1 as input value. The select pulse arrives at the ith processor with k time-units delay relative to the reference pulse, where k is the number of 1-values stored by the processors with indices smaller than i, i.e. $ps_i = k$. The relative time delay between the two pulses is detected by each processor. Thus,

Claim 6 *The n-values BPS problem can be solved on a linear APPB with n processors in a constant number of steps.* ■

Note that the BPS of n binary values can be computed in a constant number of steps on a $\sqrt{n} \times n$ [24], or on a $\log^2 n \times n$ RN [15]. If we assume the same propagation times for the signals in the two models, RN and linear APPB, the BPS takes the same time but with considerably fewer processors on the linear APPB model. Note that delay lines and the coincident pulse technique are also used in [10] to compute the *binary sum* of n binary values on a linear array with pipelined buses enhanced with switching capabilities.

5.2 Compaction

5.2.1 Compaction on a linear APPB

Given an array of values $x_0, \ldots, x_{n-1}$ of which k are nonzero, the compaction problem is the problem of moving the nonzero elements into the first k consecutive array locations. One can in addition require that the elements remain in the same order, leading to the *ordered compaction problem*. Compaction without any bound on k is harder than computing any symmetric function, and so the lower bound of $\Omega(\log n / \log\log n)$, shown for the parity function in [4], holds for any computation on a Priority-CRCW PRAM with a polynomial number of processors [31]. If k is much smaller than n, this problem can be solved in $O(\log k / \log\log n)$ time on a Common-CRCW PRAM with n processors [31].

We associate x_i with 1 if $x_i \neq 0$ and with 0 if $x_i = 0$, and apply the binary prefix sums algorithm over these values. The prefix sum value, ps_i, gives the number of 1 values stored by processors with indices smaller than i. Knowing ps_i is equivalent with knowing the position of the element stored by P_i in the compacted sequence, i.e., $ps_i + 1$. Thus, after determining ps_i, a single permutation routing suffices in order to obtain the compaction of the k values on a linear APPB. As the destination addresses of nonzero values are known and unique, the permutation routing can be implemented using a single bus cycle.

Claim 7 *The ordered compaction problem can be solved on a linear APPB with n processors in a constant number of steps.* ■

The compaction algorithm could be used when we want to apply some PRAM algorithm over a subset of keys, arbitrarily distributed on a linear APPB. In order to efficiently apply a PRAM algorithm, each processor involved should have a unique identification number and all neighboring processors should be at the same optical distance d apart. This can be obtained easily by compaction. Another useful application of the ordered compaction could be the dynamic re-scheduling of task in a parallel system.

5.2.2 2D Compaction

Let us consider an $n \times n$ AROB, and a set of k nonzero input keys v_{ij} arbitrarily dispersed among the nodes of the array. All other positions are filled with zeros. The 2D compaction problem requires moving all k values to the first $\lceil k/n \rceil$ rows of the array. In [20] this problem is solved in constant time on a reconfigurable array with $n \times n \times n$ processors (switches). We could also ask that the values be packed in a $k^{1/2} \times k^{1/2}$ mesh at, say, the lower-left corner. This problem is solved in [3] in $O(k^{1/2})$ time on a reconfigurable network with $n \times n$ processors.

We give an algorithm, which runs in a constant number of steps on an $n \times n$ AROB. Initially the AROB is configured as a 2D APPB, i.e., each row and column is structured as a linear APPB. In the first step, the BPS algorithm is used in each row. This determines, for each nonzero value, its relative position ps_{ij} in that row. The number S_i, $0 \leq S_i \leq n$, of nonzero values in each row i, $0 \leq i \leq n-1$, is also obtained. The S_i values are stored by the processors in the rightmost column.

In the second step, the prefix sums algorithm given in [24] is applied over the integer values S_i stored in the rightmost column. This can be computed in a constant number of cycles. We know now, for each row i , the number of nonzero values located in the rows below, C_i. The C_i's are broadcast in their rows. Each processor in row i which has a nonzero value, receives the C_i and then computes $F_{ij} = ps_{ij} + C_i$ which gives the possible position of v_{ij} in the compacted set. The F_{ij} values are unique for each nonzero value, v_{ij}. Processor P_{ij} computes the destination row $r_{ij} = \lfloor \frac{F_{ij}}{n} \rfloor$ and the destination column $c_{ij} = F_{ij} \bmod n$ of the value it stores. As for any two values in the same row the destination columns are different, we can use a permutation routing cycle within each row to move all nonzero values to their destination columns. Two nonzero values in the same column cannot have the same row destination, and we can apply another permutation on each column, thus moving all nonzero values to their final positions. This takes a constant number of steps.

Claim 8 *The two-dimensional compaction problem can be solved on an $n \times n$ AROB in a constant number of steps.* ■

We have an example of an algorithm for the AROB model which solves the compaction problem in the same time as RNs but with a considerably smaller number of processors (switches), that is, with a factor of $O(n)$ fewer processors. A natural open question is : Are there other problems for which this is true?

6 On the power of the APPB-like models

The simulation results obtained in this paper are significant because they allow us to compare the new models which use the optical pipelined buses for communication with the more classical one as the PRAM. In the extreme case when $\sigma_b = O(1)$, and in view of the $\Omega(\log\ n/\log\log\ n)$ lower bound on the compaction problem for the Priority-CRCW PRAM, and claims 2 and 7, we could claim that the linear APPB having n processors is more powerful than a Priority-CRCW $(n, \Theta(n))$-PRAM. In reality it seems impossible to build an APPB-like system having a constant time propagation delay for a bus of arbitrary length, and nor it is possible to implement a PRAM with constant time memory access for any number of processors, n. Therefore, care must be taken in interpreting the comparisons between the PRAM models and the arrays with optical buses. More important is the fact that, due to their ability to simulate the PRAM models, the arrays with optical pipelined buses inherit a great number of algorithms designed for the PRAM.

In order to compare the AROB structures with the RNs we give next a lower bound for the ordered compaction problem on the classical RN.

Claim 9 *The (ordered) compaction has an $\Omega(n)$ lower bound on a Linear RN with n processors. For the two-dimensional $O(n^2)$-processor RN, the lower bound on the compaction problem is also $\Omega(n)$.*

Proof: The proof of this statement is immediate. Let the linear RN be extended to a $c_1 \times n$ structure, where c_1 is constant. Consider a cut C which splits the linear RN in roughly two halves. The number of bus segments intersected by C is constant, i.e. c_1. Assume an instance of the compaction problem with $c_2 n$ nonzero values, and $c_2 \leq 1/2$ for example. These values are stored in the right-hand half and they must be compacted starting with the left-most processor in the other half of the network. Clearly, moving the nonzero values requires $\Omega(n)$ time. Similar arguments can be used to prove the lower bound of the compaction problem on a two-dimensional reconfigurable network. Actually this is true for any type of two-dimensional reconfigurable network with electrical buses because the lower bound does not depend on the interconnection capability of the switching system. ■
Taking into consideration claims 5, 8 and 9, we can say that:

Claim 10 *The two-dimensional AROB model is more powerful than the two-dimensional reconfigurable network RN.* ■

Claim 10 is true only if the end-to-end propagation delays on the two models are equal. Note that this is not a restrictive condition.

7 Conclusions

We have shown that the APPB can efficiently simulate some of the PRAM models, and that the implementation complexity of the simulation algorithms is somehow "proportional" to the power of the model being simulated. As an intermediate result it is shown that the 2D APPB is universal in the sense that can simulate efficiently any bounded degree network. An interesting problem that remains open is how to simulate efficiently and deterministically the Combining CRCW PRAM model, using either the APPB or the AROB models.

We have also shown how a 2D AROB can simulate a 2D RN. It is easy to check that these simulations are time and cost optimal (within a constant factor) as long the propagation delays on buses of the same length in the two models are considered identical. Furthermore, from a practical viewpoint, the signal propagation delay on a bus is more likely to be smaller in the case of an optical implementation. In general, the fact that in the time period given by the end-to-end signal propagation on an optical bus, up to n data can be communicated, leads us to believe that important speedups can further be obtained. We therefore seek to investigate whether similar efficient and cost optimal algorithms can be obtained for other problems using the AROB model.

Our description of different models is given at a high level, i.e., it abstracts as much as possible from technological details. In the same time, the use of electronic digital processors, fiber interconnections and opto-electronic couplers, is assumed. Some of the issues not analyzed in this paper are the synchronization of processors, clock distribution, pulse positioning, optical

fanout, etc. These problems have been investigated in [8, 27, 26]. 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 1000 processors. However, although expensive, all the opto-electronic devices assumed by our model exist. Present devices limit the operating speed to a few hundreds of MHz. It is expected that future devices could increase this speed by a factor of 1000 or more [13].

References

[1] Annexstein, F., and Baumslag, M. A unified approach to off-line permutation routing on parallel networks. *Proceedings 2nd Annual ACM Symposium on Parallel Algorithms and Architectures*. 1990, 398-406.

[2] Akl, S. G. On the power of concurrent memory access. *Proceedings of the International Conference on Computing and Information ICCI'89*. 1989, 49-55.

[3] Ben-Asher, Y., and Schuster, A. Ranking on reconfigurable networks. *Parallel Processing Letters*. **1**(2), 1991, 149-156.

[4] Beame, P., and Hastad, J. Optimal bounds for decision problems on the CRCW PRAM. *Journal of the ACM*. **36**, 1989, 643-670.

[5] Bokari, S.H. Finding maximum on an array processor with a global bus, *IEEE Transactions on Computers*, **C-33**(2), 1984, 133-139.

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

[7] Cook, S. A. A taxonomy of problems with fast parallel algorithms. *Journal of Information and Control*. **64**, 1985, 2-22.

[8] 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.

[9] Chiarulli, D. M., Melhem, R.G., and Levitan, S. P. Using coincident optical pulses for parallel memory addressing. *The Computer Journal*. Dec. 1987.

[10] Guo, Z. Sorting on array processors with pipelined buses. *Proceedings 1992 International Conference on Parallel Processing*. 1992, III 289-292.

[11] Guo, Z. Optically Interconnected Processor Arrays with Switching Capability. *Journal of Parallel and Distributed Computing*. **23**, 1994, 314-329.

[12] 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.

[13] Heuring, V. P., Jordan, H. F., and Pratt, J. P. Bit-serial architecture for optical computing. *Applied Optics*. **31**(17), 1992, 3213-3224.

[14] Harris, T. J. A Survey of PRAM Simulation Techniques. *ACM Computing Surveys*. **26**(2), June 1994, 187-206.

[15] Jang, J.-W., Park, H., Prasanna, V.K. A fast algorithm for computing a histogram on reconfigurable mesh, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **17**(2), 1995, 97-106.

[16] Levitan, S. P., Chiarulli, D. M., and Melhem, R. G. Coincident pulse technique for multiprocessor interconnection structures. *Applied Optics*. **29**(14), 1990, 2024-2033.

[17] Li, H., and Maresca, M. Polymorphic-torus network. *IEEE Transactions on Computers*. **C-38**(9), 1988, 1345-1351.

[18] Li, H., and Stout, Q. F. (eds). *Reconfigurable Massively Parallel Computers*. Prentice Hall, 1991.

[19] Leighton, F. T. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*. Morgan Kaufman Publishers, 1992.

[20] Merry, M.S., and Baker, J. W. A constant time sorting algorithm for a three-dimensional mesh and reconfigurable network. accepted by *Parallel Processing Letters*.

[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] Maresca, M., and Li, H. Connection Autonomy in SIMD Computers: A VLSI Implementation. *Journal of Parallel and Distributed Computing*. **7**, 1989, 302-320.

[23] Miller, R., Kumar, V. K. P., Reisis, D. I., and Stout Q. F. Data movement operations and applications on reconfigurable VLSI arrays. *Proceedings 1988 International Conference on Parallel Processing*. 1988, vol. I, 205-208.

[24] Olariu, S., Schwing, J. L., and Zhang, J. Integer problems on reconfigurable meshes, with application. *Proceedings 1991 Allerton Conference*. vol. 4, 1991, 821-830.

[25] Pan, Y. Order statistics on optically interconnected multiprocessor systems, *Proceedings First International Wokshop on Massively Parallel Processing using Optical Interconnections*, 1994, 162-169.

[26] Prucnal, P., Blumenthal, D., Perrier, P. Self routing photonic switching demonstration with optical control. *Optical Engineering*, **26**(5), 1987, 473-477.

[27] Qiao, C, and Melhem, R. G. Time-division communications in multiprocessor arrays. *IEEE Transactions on Computers*. **42**(5), 1993, 577-590.

[28] Qiao, C, Melhem, R. G., Chiarulli, D.M., Levitan S.P. Optical multicasting in linear arrays, *International Journal Optical Computing*, **2**, 1991, 31-48.

[29] Qiao, C. Efficient matrix operations in a reconfigurable array with spanning optical buses. to appear in *Parallel Processing Letters*.

[30] Ranade, A. G. How to Emulate Shared Memory. *Journal of Computer and Systems Sciences*. **42**, 1991, 307-324.

[31] Ragde, P. The parallel simplicity of compaction and chaining. *Journal of Algorithms*. **14**, 1993, 371-380.

[32] Wang, B. F., Chen, G. H., and Lin, F. C. Constant time sorting on a processor array with reconfigurable bus system. *Information Processing Letters*, **34**. 1990, 187-192.