Fault Tolerant Algorithms for a Linear Array with a Reconfigurable Pipelined Bus System

Anu G. Bourgeois and Jerry L. Trahan

Department of Electrical and Computer Engineering
Louisiana State University
Baton Rouge, Louisiana 70803
`{anu, trahan}@ee.lsu.edu`

Abstract. Recently, many models using reconfigurable optically pipelined buses have been proposed in the literature. All algorithms developed for these models assume that a healthy system is available. We present some fundamental algorithms that are able to tolerate up to $N/2$ faults on an N-processor LARPBS (one particular optical model). We then extend these results to apply to other algorithms in the areas of image processing and matrix operations.

1 Introduction

Currently, optical fiber is the preferred medium for telecommunication networks of long distances, due in part to its high bandwidth, reliability, low distortion, and low attenuation [5]. This wide use of optical interconnects and advances in optical and optoelectronic technologies indicate successful use of these technologies for shorter distances, such as interconnecting processors in parallel computers. The advantages of using an optically pipelined bus rather than an electrical bus to transfer information among processors are that the optical waveguides have unidirectional propagation and predictable delays. These two properties enable synchronized concurrent access to an optical bus in a pipelined fashion [15]. Combined with the abilities of the bus structure to broadcast and multicast, this architecture suits many communication-intensive applications.

As a result, several processor arrays based on pipelined optical buses have been proposed as practical parallel computing platforms [4, 13–15, 20]. Many parallel algorithms exist for arrays with pipelined buses [5, 8, 12, 16, 19], indicating that such systems are very efficient for parallel computation due to the high bandwidth available by pipelining messages. This work will focus on the *Linear Array with a Reconfigurable Pipelined Bus System* (LARPBS) [9, 19], one such model.

The number of processors involved in the systems considered raises the probability of a fault occurring. Researchers have proposed fault tolerant algorithms for many parallel architectures, such as the hypercube, mesh, and torus [2, 3, 10, 11]. They have not, however, addressed the issue of fault tolerance for reconfigurable models, and more specifically, for any of the optically pipelined models.

In this paper we present several basic fault tolerant algorithms for the LARPBS. Specifically, we have developed algorithms to calculate binary prefix sums, perform compression, sort, and perform a general permutation routing step on an N-processor array that can have up to $N/2$ static faults. We then extend these results to other fault tolerant algorithms in the areas of image processing and matrix operations.

In the next section, we describe the LARPBS and the fault model used. Section 3 explains the preprocessing phase for fault tolerant algorithms. Section 4 details the basic fault tolerant algorithms and extends the results to other more complex algorithms. Finally, in Section 5, we make some concluding remarks.

2 Model Descriptions

As mentioned in the introduction, many models exist that use an optically pipelined bus system. This work centers on the *Linear Array with a Reconfigurable Pipelined Bus System* (LARPBS), as described by Pan and Li [9]. In this section, we briefly describe this model and also explain the assumptions made for the fault model used in the following sections.

2.1 LARPBS Model

An LARPBS comprises a linear array of N processors, connected by an optically pipelined bus. Let an optically pipelined bus have the same length of fiber between consecutive processors, so propagation delays between consecutive processors are the same; we refer to this delay as one *petit cycle*. Let a *bus cycle* be the end-to-end propagation delay on the bus. We specify time complexity in terms of a step comprising one bus cycle and one local computation. For more details on the time complexity issue, see Guo *et al.* [4] and Pan and Li [9].

The optical bus of the LARPBS is composed of three waveguides, one for carrying data (the *data waveguide*) and the other two (the *reference* and *select waveguides*) for carrying address information (see Figure 1). (For simplicity, the figure omits the data waveguide, as it resembles the reference waveguide.) Each processor connects to the bus through two directional couplers, one for transmitting and the other for receiving [4, 15]. The receiving segments of the reference and data waveguides contain an extra segment of fiber of one unit pulse-length, Δ, between each pair of consecutive processors (shown as a delay loop in Figure 1). The transmitting segment of the select waveguide also has a switch-controlled conditional delay loop of length Δ between processors R_i and R_{i+1}, for each $0 \leq i \leq N-2$ (Figure 1).

To allow segmenting, the LARPBS has optical switches on the transmitting and receiving segments of each bus for each processor. With all switches set to *straight*, the bus system operates as a regular pipelined bus system. Setting the switches at R_i to *cross* segments the whole bus system into two separate pipelined bus systems, one consisting of processors $R_0, R_1, \cdots, R_i$ and the other consisting of R_{i+1}, $R_{i+2}, \cdots, R_{N-1}$. We will refer to a processor that set its

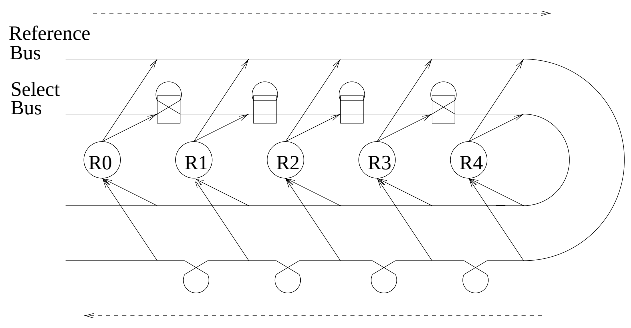

Fig. 1. Structure of a Linear Array with a Reconfigurable Pipelined Bus System (LARPBS).

segment switch to cross (the one closest to the U-turn) as the *head* of its segment. The processor furthest from the U-turn within a segment, is called the *tail* of its segment.

The LARPBS uses the *coincident pulse technique* [15] to route messages by manipulating the relative time delay of *select* and *reference* pulses on separate buses so that they will coincide only at the desired receiver. Each processor has a *select frame* of N bits (*slots*), of which it can inject a pulse into a subset of the N slots. The coincident pulse technique admits broadcasting and multicasting of a single message by appropriately introducing multiple select pulses within a select frame. When multiple messages arrive at the same processor in the same bus cycle, it receives only the first message and disregards subsequent messages that have coinciding pulses at the processor. (This corresponds to priority concurrent write.)

2.2 Fault Model

Let a *processing element* consist of a single processor, its conditional delay switches, and its directional couplers. We consider a processing element to be faulty if any one of its components is faulty, and refer to it as a faulty processor for short. Faults on any of the three optical waveguides are not considered. A fault-free processor is able to detect if either of its neighbors is faulty.

All faults that are considered are static and occur prior to the execution of any algorithm. Therefore, faults occurring during execution of a algorithm are not tolerated. The algorithms presented in Section 4 can tolerate up to $N/2$ faults on an N-processor LARPBS. These assumptions are consistent with those described by Parhami and Yeh [11].

If a conditional delay switch is faulty, that is, if it is stuck in either the cross or straight position, it remains that way for the remainder of the algorithm. Faulty segment switches are not considered, since this would result in a shorter available working array, and thus, would be a scaling problem rather than a fault tolerance problem. (For work on scalable algorithms for the LARPBS, refer to Trahan *et al.* [17, 19].)

3 Preprocessing Phase

Prior to running any algorithm on a faulty LARPBS, we perform some initial processing to ensure proper execution. Each working processor, p_i, determines the number of faulty processors p_j, where $i < j < N$, that have their conditional delay switches stuck at cross. Call the value of this suffix sum f_i. This is because any stuck delays that a select pulse travels through will alter the destination of the message sent by a working processor. By determining the total number of stuck delays ahead of it on the bus, each working processor can adjust its reference pulse to avoid miscommunication. Processor p_i shifts its reference pulse to the left by f_i slots. Then, provided each working processor has its conditional delay switch set to straight, the message sent by p_i reaches the intended destination.

First, each working processor segments the bus if it detects a faulty processor to its left. This results in two cases: 1) a segment has one or more good processors at the lower end, or 2) a segment ends with one or more faults and the head is the only good processor in the segment.

Consider the first case and one segment. The head of the segment, p_h, broadcasts its index to the segment. Any other fault-free processor, p_l, with a fault to its right, broadcasts its index to the head. The head of the segment can now determine the number of faults in the segment, call this value k. Processor p_l injects a select pulse into its highest $k/2$ slots and sends its index. Processor p_h then broadcasts a message indicating whether or not it received the message. (The segment head would receive the message if there were at most $k/2$ stuck delay switches.) If it did, p_l repeats this by injecting a select pulse into its highest $k/4$ slots. If not, then p_l injects a select pulse into slot $(N-1) - 3k/4$ through slot $(N-1) - k/2$. Repeat this process a total of $\log k$ times to determine the number of conditional delay switches that have faulted in the cross position. Worst case time complexity is when $k = N/2$, resulting in $O(\log N)$ steps.

Now consider a segment that fits the second case. The head of the segment needs the index of the head of the previous segment to determine the number of faults within its own segment. There could, however, be a string of such segments. We proceed in $\log N$ phases to relay information between these heads of segments, with each phase corresponding to one bit position of the processor indices. During phase i, where $1 \leq i \leq \log N$, each segment head with a '0' in bit position $i-1$ of its index segments the bus and and listens for those with a '1' to broadcast their index within the segment. This step is then repeated with the writers now reading, and the readers now writing. Once the preceding index is known, the segments determine the number of stuck delays within the segment, as in the first case. With $\log N$ phases, and each phase taking $O(\log N)$ steps, the total time to determine the number of stuck delay switches in each segment takes $O(\log^2 N)$ steps. This is done in $\log N$ phases rather than a simple odd/even phase, because the two processors communicating could possibly both have odd or even indices. Eventually, each segment head will receive the proper index since the two must differ in at least one bit position. In addition, the first index the segment head receives is the proper index, since the previous segment head would be segmenting the bus for each of the phases until the two communicate.

With the number of stuck delays computed for each segment, we can now determine the number of stuck delays ahead of each working processor on the array. We perform a prefix sums operation as on a tree-like structure. The head of each segment holds the data for its segment, and each other working processor holds a value of '0'. Using the indices, the processors can determine when to write and read on the bus. For each pair of processors communicating, the higher indexed processor segments the bus, in order for the two to broadcast. A working processor can determine if it is paired with a faulty processor. If the working processor is the higher indexed, then it will not receive a message from the faulty processor. If the faulty processor is the higher indexed, it will not be able to segment the bus, and the working processor will receive the index from a processor in another segment. Once a working processor determines that it is paired with a faulty processor, then the working processor continues on to the next phase. After $\log N$ phases, the head of the array broadcasts the total, so that each processor can then locally determine the number of stuck delay switches ahead of it on the bus. The prefix sum can be computed in $O(\log N)$ steps.

The next item to consider is the mapping, since each good processor will need to simulate up to two processors. We use a compaction mapping, such that the working processor with rank i simulates processors with index $2i$ and $2i+1$, for $i \leq f$, where f is the total number of faults. The remaining working processors simulate the processor with index $i+f$. In order to do this, we determine the ranks of all fault-free processors. Set the data value to '1' for each good processor and compute prefix sums in $O(\log N)$ steps. With this ranking, each working processor can determine which processor(s) it simulates. It can also properly send a message to any specific processor by adjusting its reference frame as described earlier in this section.

Combining the time to determine information on the number of stuck delay switches and to determine the mapping results provides us with the following result.

Theorem 1. *An N-processor LARPBS with up to $N/2$ faults is able to compute the number of stuck delay switches succeeding each working processor and determine the mapping of all processors to working processors in a total of $O(\log^2 N)$ preprocessing steps.*

4 Fault Tolerant Algorithms

In this section we describe some basic algorithms for an N-processor LARPBS that can tolerate up to $N/2$ faults. Using these fundamental algorithms, we can then extend the results to develop other more complex fault tolerant algorithms for the LARPBS.

Theorem 2. *Binary prefix sums of N elements can be computed on an N-processor LARPBS with up to $N/2$ faults in $O(\log N)$ steps.*

Proof. Proceed in the same manner as ranking the good processors in the previous section. In this case, however, each processor can have up to two elements to handle. With the ranking of the working processors known as well as the number of stuck delays ahead of each processor, it is possible perform the operation in $O(\log N)$ steps.

First, each good processor locally determines the total sum for the one or two elements it is simulating. Next, using the rankings of the good processors, perform prefix sums as in the tree method. (At each step, the appropriate working processors segment the bus so the corresponding processors can communicate.) Once the prefix sums is complete, each working processor can locally determine the prefix sum for each of the elements it is simulating. □

Assume that each processor of an N-processor LARPBS is either marked or unmarked based upon a local variable. Let there be x marked processors. The compression algorithm shifts all marked processors to the lower end of the array, namely processor p_0 through processor p_{x-1}.

Theorem 3. *Compression of x elements, where $x \leq N$, can be performed on an N-processor LARPBS with up to $N/2$ faults in $O(\log N)$ steps.*

Proof. First the working processors rank the marked processors, using the prefix sums algorithm of the previous theorem, in $O(\log N)$ steps. Call this the marked rank. The processor with marked rank i determines the index of the processor simulating p_i as follows and routes its data to that processor.

Let $x = 4i$. The processor, p_k, with marked rank $2i$ broadcasts its index to all processors. Next, the processor simulating processor p_{2i} broadcasts its index, j, to all processors. As a result, all processors receive the index of the processor simulating the processor with the middle rank. Next, the processor with marked rank i ($3i$) multicasts its index to $p_0, p_1, \ldots, p_{j-1}$ ($p_{j+1}, p_{j+2}, \ldots, p_{N-1}$). Similar to the previous phase, the processor simulating p_i (p_{3i}) multicasts its index to the segment of processors below (above) p_k. Repeat this phase $\log x$ times, until all ranked processors can determine the corresponding indices.

Repeat these steps for unmarked processors. These processors will determine the indices of processors starting after the last ranked processor in the previous phase, however. Once all indices of the simulating processors have been determined, send messages in two steps. First, send messages destined for an even numbered processor, then those for odd numbered processors. Routing messages this way will prevent messages from colliding at any processor. □

Theorem 4. *Sorting N k-bit integers can be performed on an N-processor LARPBS with up to $N/2$ faults in $O(k \log N)$ steps.*

Proof. We use the radix sort method and the compression algorithm to sort the N integers. The algorithm proceeds in k phases, one for each bit position of the integers. During execution of phase j, where $j \leq k$, perform compression based upon the j^{th} bit position (Theorem 3). Each phase takes $O(\log N)$ steps, for a total of $O(k \log N)$ steps. □

Table 1. Fault Tolerant LARPBS Algorithms

Algorithm	Time on Faulty	Time on Fault-Free	No. of Processors
median row	$O(\log N)$	$O(1)$	$O(N)$
image area	$O(\log^2 N)$	$O(1)$	$O(N)$
image perimeter	$O(\log^2 N)$	$O(1)$	$O(N)$
histogram	$O(\log h \log N)$	$O(\log h)$	$O(N)$
matrix transposition	$O(\log^2 N)$	$O(1)$	$O(N^2)$
matrix multiplication	$O(N \log^2 N)$	$O(N)$	$O(N^2)$

A generalized permutation routing step is one in which each processor sends at most one message and is the intended destination for at most one message.

Theorem 5. *Any generalized permutation routing step can be performed on an N-processor LARPBS with up to $N/2$ faults in $O(\log^2 N)$ steps.*

Proof. We proceed by first sorting the messages by their destinations. Since some processors may not be receiving messages, the messages are in order after the sort, but not necessarily at their final destination. So we then shift the messages to the intended processors. Perform the algorithm in two phases, one for messages destined to even numbered processors, and one for messages destined to odd numbered processors.

To perform the shifting, the processors holding the messages before the shifting determine the indices of the destination processors. Since all messages are in proper order, we can proceed in $O(\log N)$ phases broadcasting the indices of midpoints of segments, as we did for the compression algorithm (Theorem 3). The algorithm runs in $O(\log^2 N)$ steps. □

We have extended these results to apply to other algorithms in the areas of image processing and matrix operations. Table 1 lists the algorithms considered, the time complexity on a faulty and a fault-free LARPBS, and the number of processors required. The algorithms listed tolerate at most $N/2$ faults for an N-processor LARPBS. Our fault tolerant algorithms combine the techniques of the previous fundamental algorithms presented and build upon existing algorithms for the LARPBS. The image processing algorithms follow the approach of Pan and Li [9]. The matrix operation algorithms follow the approach of Li *et al.* [6, 7].

Lemma 1. *Any algorithm executed on an N-processor LARPBS with $O(1)$ faults will result in a constant factor slowdown.*

Proof. Broadcast the indices of the faulty processors in $O(1)$ steps. Each working processor locally keeps a table of the faulty processors and the working processors that are simulating them. The algorithms then run as required, with a constant number of straightforward steps to accommodate the faulty processors. □

It is important to note that the preprocessing stage presented in Section 3 is not necessary before execution of each algorithm. If there is a sequence of algorithms to be executed, the preprocessing need only be done once. Once the mapping and information on the number of stuck delays has been established, it will apply to all algorithms run thereafter on the LARPBS.

5 Conclusions

The work presented in this paper is the first to address the issue of fault tolerance for reconfigurable models, and more specifically, those with optically pipelined buses. We have shown that it is possible to design fault tolerant algorithms with low overhead.

One possible extension is to develop other complex fault-tolerant algorithms for the LARPBS, as well as minimize the time complexity for the preprocessing stage. Another possibility is to extend the results to the *Pipelined Reconfigurable Mesh* (PR-Mesh) [1,18], a multi-dimensional extension of the LARPBS, and other optically pipelined models.

References

1. A. G. Bourgeois and J. L. Trahan, "Relating Two-Dimensional Reconfigurable Meshes with Optically Pipelined Buses," to appear in *Int'l. Jour. Found. of Comp. Science.*
2. H.-L. Chen and S.-H. Hu, "Distributed Submesh Determination in Faulty Tori and Meshes," *Proc. Int'l. Par. Processing Symp.,* (1997).
3. G.-M. Chiu and S.-P. Wu, "A Fault-Tolerant Routing Strategy in Hypercube Multicomputers," *IEEE Trans. Comput.*, vol. 45, (1996), pp. 143–154.
4. Z. Guo, R. Melhem, R. Hall, D. Chiarulli, and S. Levitan, "Array Processors with Pipelined Optical Busses," *J. Parallel Distrib. Comput.*, vol. 12, (1991), pp. 269–282.
5. K. Li, Y. Pan, and S. Q. Zheng, *Parallel Computing Using Optical Interconnections*, Kluwer Academic Publishers, Boston, MA, 1998.
6. K. Li, Y. Pan, and S. Q. Zheng, "Fast and Efficient Parallel Matrix Operations Using a Linear Array with a Reconfigurable Pipelined Bus System," in *High Performance Computing Systems and Applications,* J. Schaeffer and R. Unrau, eds., Kluwer Academic Publishers, Boston, MA, 1998.
7. K. Li, Y. Pan, and S. Q. Zheng, "Fast and Efficient Parallel Matrix Multiplication Algorithms on a Linear Array with a Reconfigurable Pipelined Bus System," *IEEE Trans. Parallel Distrib. Systems*, vol. 9, (1998), pp. 705–720.
8. M. Middendorf and H. ElGindy, "Matrix Multiplication on Processor Arrays with Optical Buses," to appear in *Informatica.*
9. Y. Pan and K. Li, "Linear Array with a Reconfigurable Pipelined Bus System: Concepts and Applications," *Information Sciences – An International Journal*, vol. 106, (1998), pp. 237–258.
10. B. Parhami, "Fault Tolerance Properties of Mesh-Connected Parallel Computers with Separable Row/Column Buses," *Proc. Midwest Symp. on Cir. and Syst.,* (1993).

11. B. Parhami and C.-H. Yeh, "The Robust-Algorithm Approach to Fault Tolerance on Processor Arrays: Fault Models, Fault Diameter, and Basic Algorithms," *Proc. Int'l. Par. Processing Symp.*, (1998), pp. 742–746.
12. S. Pavel and S. G. Akl, "Matrix Operations Using Arrays with Reconfigurable Optical Buses," *Par. Algs. and Appl.*, vol. 8, (1996), pp. 223–242.
13. S. Pavel and S. G. Akl, "On the Power of Arrays with Optical Pipelined Buses," *Proc. Int'l. Conf. Par. Distr. Proc. Techniques and Appl.*, (1996), pp. 1443–1454.
14. C. Qiao, "On Designing Communication-Intensive Algorithms for a Spanning Optical Bus Based Array," *Parallel Proc. Letters*, vol. 5, (1995), pp. 499–511.
15. C. Qiao and R. Melhem, "Time-Division Optical Communications in Multiprocessor Arrays," *IEEE Trans. Comput.*, vol. 42, (1993), pp. 577–590.
16. S. Rajasekaran and S. Sahni, "Sorting, Selection and Routing on the Arrays with Reconfigurable Optical Buses," *IEEE Trans. Parallel Distrib. Systems*, vol. 8, (1997), pp. 1123–1132.
17. J. L. Trahan, A. G. Bourgeois, Y. Pan, and R. Vaidyanathan, "Optimally Scaling Permutation Routing on Reconfigurable Arrays with Optically Pipelined Buses," *Proc. 13th Int'l. Par. Process. Symp. & 10th Symp. Par. Distr. Process.*, (1999), pp. 233–237.
18. J. L. Trahan, A. G. Bourgeois, and R. Vaidyanathan, "Tighter and Broader Complexity Results for Reconfigurable Models," *Parallel Proc. Letters*, vol. 8, (1998), pp. 271–282.
19. J. L. Trahan, Y. Pan, R. Vaidyanathan, and A. G. Bourgeois, "Scalable Basic Algorithms on a Linear Array with a Reconfigurable Pipelined Bus System," *Proc. Int'l. Conf. on Parallel and Distributed Computing Systems*, (1997), pp. 564–569.
20. S. Q. Zheng and Y. Li, "Pipelined Asynchronous Time-Division Multiplexing Optical Bus," *Optical Engineering*, vol. 36, (1997), pp. 3392–3400.