SIMULATING A PR-MESH ON AN LARPBS

Mathura Gopalan[1], Anu G. Bourgeois[1], and José Alberto Fernández-Zepeda[2]

[1]*Department of Computer Science*
Georgia State University
Atlanta, GA 30302-3994
abourgeois@cs.gsu.edu

[2]*Department of Computer Science*
CICESE
Ensenada, B. C. 22860, Mexico
fernan@cicese.mx

Abstract

The unidirectional nature of propagation and predictable delays are two characteristics of optically pipelined buses that have made them popular in recent years. Many models have been proposed that use reconfigurable optically pipelined buses. In this paper we establish a relationship between a one dimensional and a two dimensional model of this type. This simulation shows that the challenge is to map the processors so that those belonging to a two-dimensional bus segment are contiguous and in the same order on the simulating one-dimensional model. We focus on the Linear Array with a Reconfigurable Pipelined Bus System (LARPBS) and its two dimensional counterpart the Pipelined Reconfigurable Mesh (PR-Mesh).

1. Introduction

Due to the properties such as unidirectional nature of propagation and predictable delays, optically pipelined buses have been gaining attention. These characteristics enable pipelining messages along a single bus and thus reduce the number of buses needed for communication. Reconfigurable architectures are capable of changing both their component structure and functionalities at every step of computation. When the reconfiguration is fast and causes little to no overhead it is referred to as Dynamic Reconfiguration [1]. The reason why dynamic reconfiguration is advantageous is because it can utilize the resources much more effectively by adapting the functionality of the hardware to the current task that has to be done.

Many different reconfigurable models using pipelined optical buses have been proposed in the literature. Some of the one dimensional models include the Linear Array with a Reconfigurable Pipelined Bus System (LARPBS) [2], the Pipelined Optical Bus (POB) [3], the Linear Array with Reconfigurable Optical Buses (LAROB) [14] and the Linear Pipelined Bus (LPB) [4]. Some of the two dimensional (or multi-dimensional) models include the Pipelined Reconfigurable Mesh (PR-Mesh) [5], the Array with Reconfigurable Optical Buses (AROB) [6], Array Processors with Pipelined Buses (APPB) [7], the Array Processors with Pipelined Buses using Switches (APPBS) [8], the Array with Synchronous Optical Switches (ASOS) [9] and the Reconfigurable Array with Spanning Optical Buses (RASOB) [10].

The commonality among the models is that they pipeline and propagate messages on a unidirectional path. The differences arise due to the presence or absence of hardware. It has already been proven that even in the presence of some physical differences, the models can still be functionally equivalent [11].

In establishing the computational complexity of the models, the translation of algorithms for models is possible. By establishing equivalence between models A and B, the algorithm developed for A can be mapped to B with a constant overhead (although a polynomial increase in the number of processors may occur). The algorithm is modified for the simulating model by using the changes that helped in establishing the equivalence between the models. If model B is a more feasible model then we have the ease of designing algorithms for model A, but have the cost and practicality of implementation on model B.

In this paper, we focus on simulating a two-dimensional PR-Mesh on a one-dimensional LARPBS. The major difference between the models is the number of bus configurations that are possible. Also, processors on the PR-Mesh may belong to multiple buses, while those on the LARPBS can only belong to one. We will show that this transformation does not increase the communication volume. Specifically, the goal of this paper is to simulate an $M \times M$ processor PR-Mesh on an N processor LARPBS where $N = M^2$.

1-4244-0054-6/06/$20.00 ©2006 IEEE

To accomplish this, we will present the simulation as a few different scenarios.

Case 1a. Each PR-Mesh processor connects to at most one bus and each bus has at most one bend. The bends signify the change in the directionality of the bus from the *x*-axis to the *y*-axis or vice versa.

Case 1b. Each PR-Mesh processor connects to at most one bus and each bus has multiple bends. The challenge in this scenario is to preserve the ordering of the processors in spite of the multiple changes in direction of each bus.

Case 2a. Each PR-Mesh processor connects to multiple buses and each bus has a single bend. When processors are connected to multiple buses the simulation becomes more complex. Therefore a processor may need to communicate with groups of processors in different subarray of the simulating LARPBS.

Case 2b. Each PR-Mesh processor connects to multiple buses and each of those buses has multiple bends.

The organization of the paper is as follows: In Section 2, we describe the LARPBS and the PR-Mesh models. In Section 3, we present an overview of the simulation of a PR-Mesh on an LARPBS. Section 4 describes the simulations for the various cases. Sections 5 and 6 provide the conclusions and future work.

2. Model Descriptions

As we are focusing on simulating the PR-Mesh on an LARPBS, the following section discusses these architectures in detail.

2.1 LARPBS Model

The *Linear Array with a Reconfigurable Pipelined Bus System* (LARPBS) [2] is a one dimensional parallel processing optical model. It is an *N*-processor array $P_1, P_2, \ldots, P_N$, linearly connected by an optical pipelined bus that makes a *U*–turn around the processors. The bus connecting the processor is assumed to have the same length of fiber between successive processors, thus propagation delays between consecutive processors are the same. A bus cycle is the end-to-end propagation delay on the bus. The time complexity of an algorithm is determined in terms of steps, where a single time step comprises one bus cycle and one local computation.

The optical bus of an LARPBS possesses three distinct waveguides. The data waveguide is used for sending data and the select and reference waveguides are used for sending address information. The top part of the bus is used for transmitting and the bottom for receiving. Each processor connects to the bus through directional couplers, one for transmitting and the other for receiving. The reference and data waveguides have an extra segment of fiber between every pair of consecutive processors on the receiving side. This segment introduces a fixed propagation delay of unit time in these two waveguides. In addition, the select bus has switch-controlled conditional delays between every pair of consecutive processors P_{i-1} and P_i on the transmitting segment of the waveguide and is controlled by processor P_i.

The coincident pulse technique helps in addressing by manipulating the relative time delay of select and reference pulses on separate buses so that they will coincide at the desired receiver. If they coincide, a double height pulse indicates to the processor to read the corresponding data frame. There is a separate set of optical segment switches present on each waveguide of the bus that help to split the LARPBS into two independent structures. If the switches at processor P_i are set, the bus is split into two separate buses, one connecting processors $P_1, P_2, \ldots P_i$ and the other connecting processors $P_{i+1}, P_{i+2}, \ldots P_n$.

2.2 PR-Mesh Model

The *Pipelined Reconfigurable Mesh* (PR-Mesh) [5] is a multi dimensional version of the LARPBS. The PR-Mesh is a *k*-dimensional mesh of processors, each having $2k$ ports. A processor can locally manipulate each of its ports so as to connect to at most one other port, to form linear buses. The PR-Mesh can appear as a directional network since both the transmitting and receiving segments are directional. The two dimensional PR-Mesh consists of an $R \times C$ mesh of processors, in which the four ports of the processors are joined to eight bus segments using the directional couplers. Figure 1 shows a detailed representation of a 2×2 PR-Mesh. The four big circles represent processors. In this figure, vertical and horizontal straight lines represent transmitting and receiving segments in both directions. The *T/R* symbol and the arrows on the lines denote the direction of transmission/reception of data. Each processor has four ports denoted by North (*N*), East (*E*), West (*W*) and South (*S)*. The directional couplers are shown in thick dark lines. Each processor controls 48 external switches (shown as small circles) used for bus interconnections. The set of external switches are divided into four quadrants (shown in dotted rectangle boxes for the bottom right processor). The black switches labeled *E1, E2, W1, and W2* enable forming

row buses and switches *N1, N2, S1,* and *S2* enable forming column buses, for each of the quadrants. The white switches labeled *F1, F2, F3, F4* for each quadrant allow forming two dimensional buses that run in two directions, for example North-West, South-East and so on. Essentially, they enable fusing horizontal and vertical segments together so that the bus can bend from the *x*-axis to the *y*-axis or vice versa in any direction.

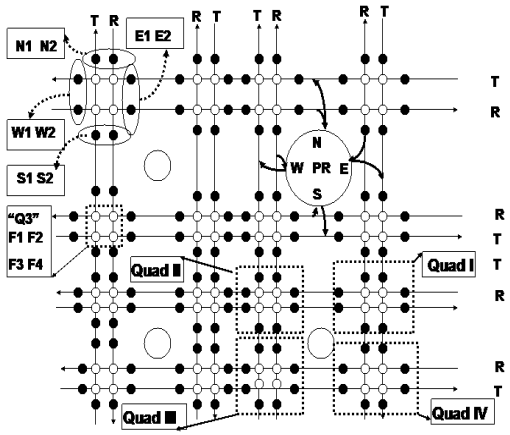

Figure 1 The structure of a PR-Mesh.

Processors can function as a disconnected processor, head (one closest to the *U*-turn), tail (one farthest from the *U*-turn) or simply an intermediate processor (processors that are neither head nor tail) on a bus. Figure 2 represents the possible roles of the processors (Black circles represent switches that are set to fuse connections and intersections without circles have open connections.) Figure 3 shows the corresponding PR-Mesh for Figure 2 processors. A PR-Mesh processor can be the head of the segment for up to two directional buses. The head of a bus will set some of its F^* ports (where F^* represents any of its fusing ports *F1*, *F2*, *F3* and *F4*) in any of its quadrants. This fusing causes a *U*-turn in the bus, thereby connecting the transmitting to the receiving segment. Processors on a column bus fuse their *N1, N2, S1* and *S2* ports. Fusing vertical segments together in the same direction makes the bus continue along the same vertical direction. Processors may be on one or both of its column buses. Processors that are the tail of a bus fuse switches in only one quadrant thereby stopping the progress of the bus along the same direction. Processors on a row bus fuse their *E1, E2, W1* and *W2* ports making the bus progress in the same horizontal direction.

Processors that are not connected to any bus open all their switches. Since the PR-Mesh is a two dimensional model, it has the capability to change directions of a bus from horizontal to vertical or vice versa. This is called a bend in the bus. To achieve a bend, the diagonally opposite ports in the same quadrant are fused. A *pivot-processor* is the processor at which the bus bends and fuses a horizontal to a vertical segment. A bus can bend in eight possible ways.

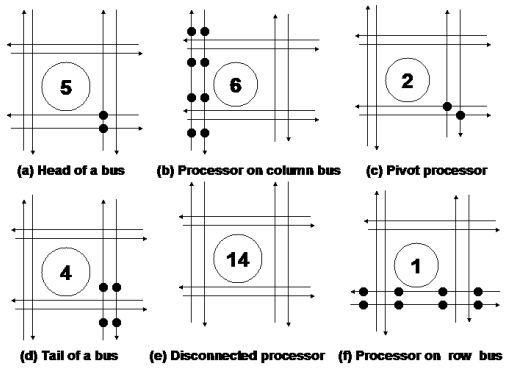

Figure 2 Roles of PR-Mesh Processors

3. Overview of the Simulation

Before presenting the simulation of the PR-Mesh on a LARPBS, we describe how to map processors from a PR-Mesh to an LARPBS model. For this, a simple row-major arrangement of processors on the PR-Mesh based on their indices is sufficient for the initial linear arrangement of processors in the LARPBS. We now discuss some of the main procedures of the simulation.

Identification and Ranking of Components: A component is a set of processor ports that are connected to the same bus on the PR-Mesh. The simulation is performed for a two dimensional PR-Mesh and hence buses can be on *x*-axis alone or *y*-axis alone or can span both axis. The idea is to rank all the components and map each of them as a separate sub-array in the LARPBS. The algorithm treats processors that do not belong to any bus as a single component.

Identification of Component-Members*:* The component-members of a specific component refer to the set of processors that have at least one port that belongs to that component. Since each processor on the PR-Mesh can have ports connected to multiple buses, the simulation algorithm must determine which ports belong to which components.

Ranking Component-Members: This procedure ranks the processors for each specific component. This will allow

us to map processors of a bus from the PR-Mesh to a set of contiguous processors in the LARPBS.

Switch and Port Configurations of Component Members: As the processors are mapped from the PR-Mesh to LARPBS in row major order, the processors retain their port as well as switch configurations. For example, processors with delay switches in cross position will retain that configuration.

4. Simulation of PR-Mesh on LARPBS

We now present the simulation by considering simpler to more complex bus configuration patterns. The four cases are as described at the end of Section 1.

4.1 CASE - 1(a)

This case involves simulating an $M \times M$ processor PR-Mesh on an N-processor LARPBS, where $N=M^2$. Our assumptions are that each PR-Mesh processor can be connected to only one bus and that bus bends only once. The main aim here is to successfully identify and rank different buses as well as the processors that appear on each bus. Then perform a one-to-one mapping between PR-Mesh and LARPBS processors. We describe the pseudocode of the simulation below.

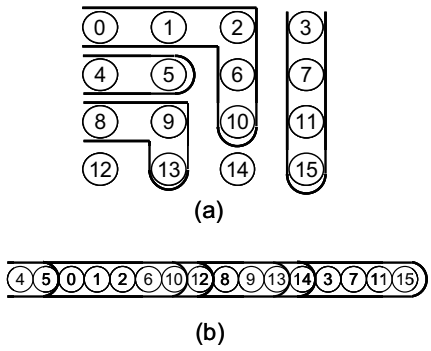

Figure 3 Simulation of PR-Mesh on LARPBS

Figure 3 (a) shows the bus configuration on the PR-Mesh and (b) shows the final assignment of the PR-Mesh processors on the LARPBS. The first step in the algorithm is to rank the buses that exist on the PR-Mesh.

We achieve this by computing a prefix sum on the heads and disconnected processors.

In the next few stages, the algorithm identifies the processors that lie across the same bus. First, the algorithm finds processors that lie on the horizontal part of the bus and then temporarily ranks them.

This is done by arranging processors in row major order. All processors whose East-West ports are not connected set their segment switches. Each processor who segmented in the previous step sends their index to its left/right neighbor processor. The left neighbor is the processor sending its index P_i–1, and right neighbor is processor sending its index P_i+1. All disconnected processors also do the same.

```
Begin
  Perform Bus Ranking
      Compress heads of segments and disconnected
processors
      Compute the prefix sum on these processors
   Identify Row Segments
            Arrange processors in row major order
            Group processors lying on same bus
            Rank processors in along row segments
          Pivot nodes hold total number of processors
   Identify Column Segments
            Arrange processors in column major order
            Group processors lying on same bus
            Rank processors in along column segments
            Pivot nodes hold total number of processors
Re-Rank Processors
     If pivot node gets bus rank from column segment
            Processors in the column segment retain rank
            Processor in row segments adjust ranks
     If pivot node gets bus rank from row segment
            Processors in the row segment retain rank
            Processor in column segments adjust ranks
 Compute Slot start value
     Compress heads of segments and disconnected
processors
     Compute prefix sum on total number of processors
    Broadcast slot start values to all processors on the bus
      Each processor compute new index
      Arrange each processor based on new index
End
```

Pseudocode for CASE 1-(a)

If a processor receives a message, it sets its segment switch. A receiving processor now knows that the processor on its left/right had set its segment switch since it did not lie on a row bus, therefore it cannot also lie on the row bus that the receiving processor lies on. Same is done for recognizing processors along column segment but the indices of left and right neighbor processors are $P_i - M$ and $P_i + M$. This is a temporary ranking as neither remaining processors along the bus nor the direction of the head processor is known. Once the horizontal segments of the processors are identified by arranging processors in row-major format, the pivot processor now knows the number of processors along a row segment and if the head of the segment lies in this segment.

Processors along the column of the bus can be found by arranging processors in column-major order and finding the number of processors along this segment. The ranks have to be readjusted as the direction of the bus is now known. This is done by rearranging the processors in row major order and

broadcasting the number of processors that are present between the head of the segment and pivot processors so processors along the pivot-tail of the segment need to re-adjust their ranks. The final step of the algorithm is finding which slot is to be occupied by which bus by computing the prefix sum along each bus as they are ranked.

Complexity Analysis:

We now describe the complexity analysis of the simulation algorithm. Compression algorithm takes $O(1)$ time [2]. For processors to compute their temporary ranks along a row or column bus takes $O(1)$ time. Arranging the processors in row major order as well as column major order to identify processors along each row and column segment takes $O(1)$ time [2]. All communication between processors takes $O(1)$ time. All the internal functions that the processors perform, for example adjusting their ranks once the other processors along the bus have been identified, finding number of processors along their segments, etc. takes $O(1)$ time. The prefix sum is computed for the head of the segments. For integers with bounded magnitude, the algorithm for prefix sum computation takes $O(\log \log N)$ time using N processors [12]. Permutation routing over the LARPBS takes $O(1)$ time [11].

Hence the complexity of the algorithm is $O(\log \log N)$. Note that the efficiency of the simulation lies in the prefix sum computation.

4.2 CASE 1- (b)

Our assumptions for this case are that each PR-Mesh processor is connected to only one bus and each bus bends more than once. Figure 4 shows an example of this assumption. The initial steps of the simulation are the same as the previous section. The ranking of processors is different since there are many bends in the bus. We provide a detailed explanation of the prefix sum computation using a binary tree-like method. Note that after forming row and column segments, each pivot processor becomes aware of the number of processors ahead and behind it and also of the next pivot processor that it might need to communicate with in order to find the number of processors in that segment.

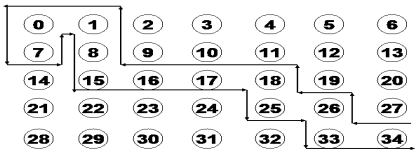

Figure 4 Buses with Multiple Bends

We now need to determine the directionality of the bus and pass this on to other processors. Pivot processors are the processors that achieve this task. Ranking the processors in their segments can be done only after learning the number of processors ahead of them. We describe the pseudocode of the simulation below.

```
Begin
  Perform Bus Ranking
      Compress heads of segments and disconnected
          processors.
      Compute the prefix sum on these processors
  Identify Row Segments
      Arrange processors in row major order
      Group processors lying on same bus
      Pivot nodes hold total number of processors in
          row segment
  Identify Column Segments
      Arrange processors in column major order
      Group processors lying on same bus
      Pivot nodes hold total number of processors in
          column segment
  Rank Processors
    Repeat on pivot nodes until prefix sum is computed
          {
          Perform ranking using binary prefix sum
          algorithm
          Pivot processor send index to pivot ahead of
           it
          Pivot receiving index send prefix sum
          Pivot receiving index also send next pivot index
          Pivots newly learning index of head of
           segment
          Send their index to head
          }
          After ranking tail send rank to head
  Compute Slot start value
      Compress heads of segments and disconnected
processors
      Compute prefix sum on total number of processors
      Broadcast slot start value to all pivots on the bus
      Pivots broadcast slot start to processors in their
segment
      Each processor compute new index
    Arrange each processor based on new index
End
```

Pseudocode for CASE 1-(b)

Ranking now becomes a prefix sum computation on the number of processors held in each segment. The traditional binary tree method cannot be used here for

the prefix sum computation because the indices of the processors are not structured. In other words, processors are not necessarily communicating with those spaced $2i$ apart, so the processors are unaware of who to communicate with in the next stage of the computation. Hence, all the segment switches are now set straight. During the formation of row/column segments each pivot node must provide the index as well as the sum computed so far to the processor communicating with it.

As a final step, the communicating processor also provides the id of the next pivot processor to communicate with. This is continued until the prefix sums are computed. In addition, each pivot node that newly learns the identity of the head of the segment must send its index to the head of the bus.

The head of the segment becomes aware of all the pivot nodes at the end of the prefix sum computation. This becomes vital because after the processors are ranked, the next step is to find the slot that this bus needs to occupy depending on the rank. This is continued until the last step in which the final nodes posses the prefix sum of all the pivot nodes in front of it. Prefix sum is then used compute the rank of the other processors on the same bus. Once the ranking is completed, the rest of simulation is similar to the previous section.

Complexity: While the rest of the simulation is similar to the previous case, the complexity has increased due to ranking the processors by the binary tree like method. The complexity is now $O(\log \log N + \log b)$ where b denotes the number of bends in the bus. It is also prudent at this stage to compute the worst case value of b. The architecture of the PR-Mesh allows the buses to bend at every opportunity and form a meandering structure. The bus can be bent twice by a single processor. Hence the number of bends is bounded by $O(N^2)$ bends. But it should be noted that it is highly unlikely that the bus is bent so many times. The number of bends will typically be much less than the worst case as defined. In simpler terms, $b << O(N^2)$. Hence the worst case complexity of simulating a PR-Mesh model on an LARPBS is $O(\log \log N + \log N)$ for case1(b).

Overview of Simulation Involving Multiple Buses: Simulating processors that are on multiple buses is much more complicated due to the fact that within a single bus cycle, a processor might have to function as members of different sub arrays in the LAPRBS which is not possible. There has to be an increase in the number of processors so that they can be accommodated on as many as four buses.

The approach of allowing an increase in the number of processors in the simulating model was utilized in the simulation of the Cycle Free Linear Reconfigurable Mesh (CFLR-Mesh) by the LR-Mesh [13-14]. In the simulation of a two dimensional PR-Mesh on an LARPBS the increase in the number of processors is constant instead of a polynomial increase.

We introduce a notation prior to presenting the results of the simulation [14]. For a model Z, let $F= Z (T, Constant (N))$ denote the class of problems solved by the model Z in $O(T)$ steps with a constant increase in the number of processors. Here the LARPBS is the model represented by Z and the two parameters of F are to be found. From the configuration of the PR-Mesh, it is known that at the most, each processor can be on four buses and hence the value of the constant is four. The equation is now modified as $F= Z (T, 4 (N))$. Next the time needed for the simulation is to be computed.

Preprocessing Step: The first step in the simulation is the indexing of processors and then arrangement or mapping on the LARPBS. The four copies of processor P_i have indices P_{ia}, P_{ib}, P_{ic} and P_{id}, respectively. Processor with index P_{ia}, is deemed as the "*master processor*" and holds the port and switch configurations of Processor P_i. The other three processors are the slave processors at the beginning of the simulation. All processors with indices P_{ix} are sorted in ascending lexicographic order.

During pre-processing, if processor P_{ia} determines it is connected to multiple buses, then processor P_{ia} simulates the bus segment with transmission from east to west. It makes processor P_{ib} simulate the bus segment with transmission from west to east, P_{ic} bus segment with transmission from north to south, and P_{id} bus segment with transmission from south to north.

The respective port and switch configurations are passed on to these processors in constant time. After this step, the processors can independently and need not pass on any information to the master processor. An additional point to be noted is that when a processor is a head of multiple buses those buses should be ranked consecutively. At this point, we have separated each processor that is on multiple buses into independent segments.

4.3 CASE 2 - (a)

Here the main problem involves the elimination of duplicate processors that are present in certain

segments. We first identify row and column segments of buses by arranging all P_{ia} processors together in increasing order of i such that they form *Group-a*. Form *Group-b*, *Group-c* and *Group-d* similarly in row major order. Processors in *Group-c* and *Group-d* are then arranged in column major format. The processors along row or column buses are identified using the same procedure as in previous cases. In order to perform ranking, all processors straighten their segment switches to form a single LARPBS.

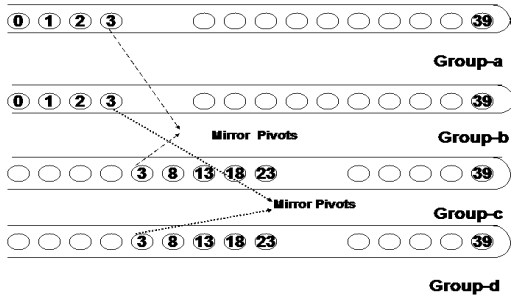

Figure 5 Mirror Pivots Elimination

Since the bus has only one bend, the bus is divided into two parts with a mirror image of the same pivot processor in both the segments as seen in Figure 5. One of the two mirror pivots will be in a segment where the identity of the head of the segment is known. So the pivot processor that knows the identity of the head of the segment and rank of the bus contacts the mirror pivot to rank processors in the other segment.

Note that all processors know the id as well as the index (when arranged in terms of groups) of the processor simulating its pivot. Thus multiple one-to-one communications can take place in a single step. Only one of the two mirror pivot processors is ranked and it is the communicating processor while the other one becomes idle after passing on the information.

The adjustment of the ranks is similar to case 1(a). After this step, all disconnected and idle processors are ranked and pushed to one end of the LARPBS. The rest of the simulation proceeds as case 1(a). The time complexity for this case is the same as case 1(a).

4.4 CASE 2 - (b)

We can summarize the problem of simulating this scenario into two main steps. Namely, the identification and elimination of mirror pivots to concatenate the separate bus segments into one and ranking processors on the bus. After completing the fist step the processors are now on single bus with multiple bends the rest of the simulation is done as discussed in Case 1(b).

```
Begin
  Perform Bus Ranking
      Compress heads of a segments and disconnected
      processors.
      Compute the prefix sum on these processors
  Identify Row Segments
      Arrange processors in row major order
      Group processors lying on same bus
      Rank processors in along row segments
      Pivot nodes hold total number of processors in row
      segment
  Identify Column Segments
      Arrange processors in column major order
      Group processors lying on same bus
      Rank processors in along column segments
      Pivot nodes hold total number of processors in column
      segment
Elimination of Mirror Pivots
      If pivot node gets bus rank from column segment
        Preserve and rank pivot in column segment
        Eliminate pivot in row segment
      If pivot node gets bus rank from row segment
        Preserve and rank pivot in row segment
        Eliminate pivot in column segment
Re-Rank Processors
    If pivot node gets bus rank from column segment
      Processors in the column segment retain rank
      Processor in row segments adjust ranks
    If pivot node gets bus rank from row segment
      Processors in the row segment retain rank
      Processor in column segments adjust ranks
Compute Slot start value
    Compress heads of segments and disconnected
      processors
    Compute prefix sum on total number of processors
    Broadcast slot start values to all processors on the bus
    Each processor compute new index
    Arrange each processor based on new index
End
```

Pseudocode for CASE 2-(a)

The first step is similar to the previous case except that there is more than one pair of such mirror pivots. To eliminate all, the processors straighten their segment switches to form a single LARPBS. The pivot processor that knows the identity of the head of the segment and rank of the bus contacts the mirror pivot to rank processors in the other segment. Since the direction of the transmission is known, the mirror-pivot farther in the direction of transmission is always chosen for ranking and the eliminated pivot informs the chosen one of the number of processors in its segment and also the identity of the pivot that it needs to communicate with in the next steps. After this step all the segments have been joined together and now the simulation for the ranking is similar to Case 1(b). Once ranking has been completed, the rest of the simulation is the same as Case 2(a) for ranking the idle and disconnected processors and finally computing the

slots to be occupied. The time complexity for this case is the same as case 1(b).

5. Conclusions

From the simulation it is established that a two dimensional $M \times M$ PR-Mesh can be simulated on an N or $4N$ processor LARPBS (where $N = M^2$). The cases considered for the simulation differ based on the varying complexity of bus structures. Since the number of processors needed for the simulation differ based on the complexity of the bus structure and so does the time taken to perform the simulation, choosing an appropriate case will yield better and efficient simulation performance.

This simulation is first of its kind to establish a relationship between a one dimensional and a two dimensional optical model. It is also shown that the move in fact, has caused no overhead in the volume of communication. The complexity in reconfigurable architecture is either due to the functionalities provided by the models or the complexity of the bus structure. In this case the functionalities provided by both models are the same. The complexity of the PR-Mesh is due to the latter aspect. In order to handle the bus complexity, the number of processors was increased. This is due to the fact that each bus is represented as a separate sub-array in the LARPBS, so a processor that is a part of multiple buses may have to communicate with processors in different subarray with in a single bus cycle.

6. Future Work

The PR-Mesh is a two dimensional extension of the LARPBS hence the natural correspondence between them was exploited. However, there are many other models that have much richer switch and port configurations or functionalities that they provide. Hence there should be attempts to study the relationships of these models with respect to the LARPBS as well as their one dimensional counterparts. As the PR-Mesh is a k-dimensional model, it would be prudent to develop a more generalized algorithm for any value of k. With an increase in the number of dimensions the complexity of the bus structure will increase. Some areas of concern are the mapping from different dimensions of the PR-Mesh to the LARPBS, the placement of ports and how the processors on different dimensions are connected. Similar to this simulation, the identification of different buses, ranking of the buses, identification, ranking of the processors on the different buses needs to be determined.

Our simulation provides us with a better understanding of the overhead required for simulating the PR-Mesh on the LARPBS. The overhead involved in the simulation is mainly due to the increase in the number of processors.

Thus in simulations involving higher dimensions though a constant or a polynomial increase in the number of processors is permissible, it would be a challenge to keep the number of processors the same as the simulated model and investigate the corresponding time complexity.

References:

[1] R. Vaidynathan and J. L. Trahan, *"*Dynamic Reconfiguration: Architectures and Algorithms*"*, Kluwer Pub., 2003.

[2] Y. Pan and K. Li, "Linear array with a reconfigurable pipelined bus system: Concepts and applications", *Inform. Sci.* vol. 106, (1998), 237-258.

[3] S. Q. Zheng and Y. Li, "Pipelined asynchronous time-division multiplexing optical bus", *Opt. Eng*. vol. 36, (1997), 3392-3400.

[4] Y. Pan, "Order statistics on optically interconnected multiprocessor systems", *Opt. Laser Tech*. vol. 26, (1994), 281-287.

[5] A. G. Bourgeois and J. L. Trahan, "Relating Two-Dimensional Reconfigurable Meshes with Optically Pipelined Buses," *International Journal on Foundations of Computer Science,* vol. 11, (2000), pp. 553-571.

[6] 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.

[7] M. Middendorf and H. ElGindy, "Matrix Multiplication on Processor Arrays with Optical Buses", *Informatica*, vol. 22, no. 3, (1998).

[8] Z. Guo, "Optically Interconnected Processor Arrays with Switching Capability", *Journal of Parallel and Distributed Computing* vol. 23, (1994), pp. 314-329.

[9] C. Qiao and R. Melhem, "Time-Division Optical Communicationsin Multiprocessor Arrays", *IEEE Trans. Comput*., vol. 42, (1993), pp. 577-590.

[10] C. Qiao, "On Designing Communication-Intensive Algorithms for a Spanning Optical Bus Based Array", *Parallel Processing Letters*, vol. 5, (1995), pp. 499-511.

[11] J. L. Trahan, A. G. Bourgeois, Y. Pan, and R. Vaidyanathan, "An Optimal and Scalable Algorithm for Permutation Routing on Reconfigurable Linear Arrays with Optically Pipelined Buses", *Journal of Parallel and Distributed Computing,* vol. 60, (2000), pp. 1125-1136.

[12] A. Datta, "Multiple Addition and Prefix Sum on a Linear Array with a Reconfigurable Pipelined Bus System", *The Journal of Supercomputing*, vol. 29, (2004), pp. 303–317.

[13] J. L. Trahan, A. G. Bourgeois, and R. Vaidyanathan, "Tighter and Broader Complexity Results for Reconfigurable Models", *Parallel Proccessing Letters*, vol. 8, (1998), pp. 271-282.

[14] S.Pavel and S.G Akl, "Integer Sorting and routing in arrays with reconfigurable optical buses", *Proceedings of International Conference of Parallel Processing,* pp. III-90-III-94, 1996.