1-12-2006

Simulating a Pipelined Reconfigurable Mesh on a Linear Array with a Reconfigurable Pipelined Bus System

Mathura Gopalan

SIMULATING A PIPELINED RECONFIGURABLE MESH ON A LINEAR ARRAY WITH A RECONFIGURABLE PIPELINED BUS SYSTEM

by

MATHURA GOPALAN

Under the Direction of (Anu Bourgeois)

ABSTRACT

Due to the unidirectional nature of propagation and predictable delays, optically pipelined buses have been gaining more attention. There have been many models that have been proposed over time that use reconfigurable optically pipelined buses which in turn function based on numerous parallel algorithms. These models are well suited for parallel processing due to the high bandwidth available by pipelining of messages. The reconfigurable nature of the models makes them capable of changing their component's functionalities and structure that connects the components at every step of computation. There are both one dimensional as well as k – dimensional models that have been proposed in the literature. Though equivalence between various one dimensional models and equivalence between different two dimensional models had been established, so far there has not been any attempt to explore the relationship between a one dimensional model and a two dimensional model.

The aim of this thesis is to establish a relationship between a one dimensional and a two dimensional model. This simulation will be a first of its kind. It will show that a move from one to two or more dimensions does not cause any increase in the volume of communication between the processors as they communicate in a pipelined manner on the same optical bus. When moving from two dimensions to one dimension, 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 one-dimensional model. This does not increase any increase in communication overhead as the processors instead of communicating on two dimensional buses now communicate on a linear one dimensional bus structure.

Hence a very commonly used model Linear Array with a Reconfigurable Pipelined Bus System (LARPBS) and its two dimensional counterpart Pipelined Reconfigurable Mesh (PR-Mesh) are chosen to understand the relationship between one dimensional and two dimensional models. Since the PR-Mesh does not allow buses to form cycles, it is feasible to study its functionality with respect to the LARPBS. In this thesis an attempt has been made to present a simulation of a two dimensional PR-Mesh on a one dimensional LARPBS to establish complexity of the models with respect to one another, and to determine the efficiency with which the LARPBS can simulate the PR-Mesh.

For the simulation, instead of taking the most likely scenario in which processors are connected to multiple buses and the buses having a much more complex structure, we have considered different scenarios. These scenarios are based on the varying complexity of bus structures. It is possible that the number of processors needed for the simulation increase or decrease based on the complexity of the bus structure and so does the time taken to perform the simulation. Hence it is pertinent to analyze every possible scenario so that the simulation performance can be enhanced.

INDEX WORDS: LARPBS, PR-Mesh, Equivalence, Simulation, Optical models

SIMULATING A PIPELINED RECONFIGURABLE MESH ON A LINEAR ARRAY WITH A RECONFIGURABLE PIPELINED BUS SYSTEM

by

MATHURA GOPALAN

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of

[**Master of Science**]

Georgia State University

SIMULATING A PIPELINED RECONFIGURABLE MESH ON A LINEAR ARRAY WITH A RECONFIGURABLE PIPELINED BUS SYSTEM

by

MATHURA GOPALAN

Major Professor: Dr. Anu G. Bourgeois
Committee: Dr. Yi Pan
Dr. Michael Weeks

Electronic Version Approved:

Office of Graduate Studies
College of Arts and Sciences
Georgia State University
December 2005

ACKNOWLEDGEMENTS

I would like to express my gratitude to my advisor, Dr. Anu G. Bourgeois for her encouragement, advice and guidance throughout my thesis work. My special thanks to both the committee members, Dr. Yi Pan and Dr. Michael Weeks for rendering their time in reviewing this thesis report. Also, I would like to thank my husband Kaushik Kapisthalam and my parents Mr. T. Gopalan and Mrs. Usha Gopalan for their love and support.

TABLE OF CONTENTS

List of Tables

List of Figures

List of Abbreviations

PR-Mesh	-	Pipelined Reconfigurable Mesh
LARPBS	-	Linear Array with a Reconfigurable Pipelined Bus System
POB	-	Pipelined Optical Bus
LAROB	-	Linear Array with Reconfigurable Optical Buses
LPB	-	Linear Pipelined Bus
AROB	-	Array with Reconfigurable Optical Buses
APPB	-	Array Processors with Pipelined Buses
APPBS	-	Array Processors with Pipelined Buses using Switches
ASOS	-	Array with Synchronous Optical Switches
RASOB	-	Reconfigurable Array with Spanning Optical Buses
LR-Mesh	-	Linear Reconfigurable Mesh

1 INTRODUCTION

The advancement in the optoelectronic technologies has caused increase in the usage of optical interconnects and thus optical computing has emerged as a new computing field. The optical bus is one such example. Due to the advantages like unidirectional nature of propagation and predictable delays, optically pipelined buses have been gaining attention. While the unidirectional nature of the propagation helps in pipelining of messages where multiple messages are in transit along the same bus there by reducing the number of buses needed for communication; the predictable delays are advantageous in two ways. First they allow pipelining of messages; in the sense that multiple messages can travel at the same time on the bus. The second advantage is the introduction of limited delays which are helpful during the addressing. It should be noted that because of these features, a synchronized, concurrent access to an optical bus in a pipelined fashion is possible. The bus has the capability to broadcast and multicast information with much more efficiency than with electrical buses thus making the architecture with optically pipelined buses suitable for many parallel processing systems. The success of an application lies in the fact of how well the processors have been utilized which in turn depends on how good the communication between processors is. Many models that have been proposed over time that employ pipelined optical buses which in turn function based on numerous parallel algorithms. This indicates that these models are well suited for parallel processing due to the high bandwidth available by pipelining of messages [1].

Many optical models are designed as optical *reconfigurable* models. Reconfigurable models are capable of changing their component's functionalities and structure that connects the components at every step of computation. Thus the reconfigurable architectures are capable of changing both their component structure and functionalities at each and every step of

computation. For example the reconfigurable models can use the bus as a computation tool for different problems at hand. When the reconfiguration is fast and causes little to no overhead it is termed as *Dynamic Reconfiguration*. It can be said that a dynamically reconfigurable architecture comprises a large number of computing elements (such as processors) that are connected by a reconfigurable medium (such as an optical bus) that is used for communication purposes [1]. The processors used in these kinds of architecture are assumed to have a local memory of their own. There is no shared memory concept here. These processors function synchronously in a single instruction multiple data (SIMD) architecture [1]. In an SIMD environment all active processor work on the same instruction while the data on which they are doing this operation might differ. The communication among processors takes place via the optical bus. 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. In other words, it describes the adaptability of the hardware to take advantage of a problem instance. Dynamic reconfiguration envisions greater speed and efficiency in computations. Hence this has promoted a great amount of interest among many researchers and dynamic reconfiguration had emerged as a powerful computing paradigm.

There have been both one dimensional as well as multidimensional models that have been proposed. 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) [15] and the Linear Pipelined Bus (LPB) [4]. Some of the two 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] . Refer to the Appendix for the model architectures.

The commonality among all of these models is that they pipeline messages and propagate them on a unidirectional path. Also, most of the models allow processors that are in the downstream path of the message being sent to affect the destination of the messages. In simpler terms, the actual destination may or may not be a selected destination that was chosen originally. On the other hand, the differences arise due to some functionality, such as the placement or presence of segment switches, delay loops, etc. that may or may not be present in a model. It has been proven already that even in the presence of some physical differences, the models still can be functionally equivalent [11]. For example the LARPBS, which possesses segmentation capability, and LPB, which lacks the segmentation capability, are able to perform the same algorithm of computing the prefix sum in constant time. In fact it has been proven that any problem that can be solved by an LARPBS can also be solved by an LPB using the same number of processors in same amount of time [11]. The idea is further explained in forthcoming sections. In investigating the computational powers of these models, one of the factors considered is how well a certain model performs as against some other model or how well a resource is utilized by a model with respect to another one [1]. For example, the Array Processors with Pipelined Buses using Switches (APPBS) permits processors to change switch configurations between bus cycles, after each bus cycle or a once or twice during a *petit* cycle [This denotes the delay between a processor and its adjacent neighbor]. Hence it is capable of generating many more configurations than other models, thereby exhibiting a much higher degree of reconfiguration. When trying to simulate this model on another two dimensional model, the number of processors will have to increase to accommodate all the possible bus configurations of the APPBS. Hence it becomes

vital to study and understand the computational power, capability and equivalence of the reconfigurable pipelined optical models with respect to one another. These theoretical studies also help in strengthening the usefulness of features in models and can make us understand when and where to use each one of them. On relating two models say model *A* and model *B* it can be studied what model *A* can do given certain resource that model *B* can't do. Or how much more resource will model *B* need to do the same amount of work done by model *A* and so on.

In establishing the computational capacity of the models, the translation of algorithms for models is possible. Sometimes algorithms are easy to develop for certain models, say model *A* when compared to a model *B*. By establishing equivalence between models *A* and *B*, the algorithm developed for *A* can be mapped to model *B*. The algorithm is modified for the other model by using the changes that helped in establishing the equivalence between the models. This procedure of mapping algorithms is known as the translation of algorithms. These studies also help in mapping the resources required for a problem to be solved on two different models once their computational powers with respect to each other are known. A benefit of doing this is 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 implementing on model *B*.

It is to be noted that though equivalence between various one dimensional models and various two dimensional models had been established, so far there has not been any attempt to explore the relationship between a one dimensional model and a two dimensional model. The main aim of the thesis is to relate a one dimensional model and a two dimensional model. This provides a frame work for a first ever simulation of a two dimensional model on a one dimensional model that will provide a basis to understand the computational powers of the model with respect to each other. The idea for the research comes from the point that, a move from a

one to two or more dimensions does not cause any increase in the volume of communication as the processors communicate in a pipelined manner on the same optical bus. In forthcoming sections will establish that the PR-Mesh does not allow cycles to be formed in the bus configurations. Hence if a bus runs across *x*-axis and *y*-axis, it implicitly states that a processor can be on this bus only once in the same direction. Hence when trying to move from two dimensions to one dimension, it mainly involves moving the processors from the *y*-axis to *x*-axis in the order that it appears on the two dimensional bus. Therefore it can be seen that this does not increase the communication volume. The major difference lies in the fact that due to the many more bus configurations that are possible, capabilities of a model may increase. Hence a very commonly used model, the LARPBS and its two dimensional counterpart, the PR-Mesh are chosen to understand the relationship between one dimensional and two dimensional models. Since the PR-Mesh does not allow buses to form cycles, it is feasible to study its functionality with respect to the LARPBS.

Thus the goal of the thesis is to simulate an M x M PR-Mesh on an N processor LARPBS where $N = M$ x M. To accomplish this, we will present the simulations as a few different scenarios. First we will consider simulating a PR-Mesh such that each processor is connected to at most one bus and the 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. The bends are indicative of the fact that the buses are no longer linear. Hence one bend would indicate that the bus changes direction from the *x*-axis to the *y*-axis or from the *y*-axis to the *x*-axis only once. Next we will consider simulating a PR-Mesh such that each processor is connected to at most one bus and the bus had multiple bends. Multiple bends indicate that the directionality of the bus changes many times. The challenge is to be able to preserve the ordering of the processors when there are

multiple bends. This is explained in detail in the forthcoming sections. Third case will involve simulating a PR-Mesh such that each processor is connected to multiple buses and the buses have a single bend. When processors are connected to multiple buses the complexity of the model increases. Therefore a processor may need to communicate with groups of processors in different sub-arrays of the LARPBS. Our final case involves simulating a PR-Mesh such that each processor is connected to multiple buses and each of those buses has multiple bends. We will also analyze and present the complexity of the simulation algorithms.

For the simulation, instead of simply analyzing only the most likely and probable scenario in which processors are connected to multiple buses and the buses have a complex structure, we have considered different scenarios. These scenarios differ based on the varying complexity of bus structures. It is possible that the number of processors needed for the simulation increase or decrease based on the complexity of the bus structure and so does the time taken to perform the simulation. Hence it is prudent to analyze every possible scenario so that the simulation performance can be enhanced.

In Section 2 the model descriptions of the LARPBS and the PR-Mesh provide a basis for understanding the architectures, features, some basic algorithms and finally the complexity of each model. In Section 3 some of the background works relating various one dimensional and two dimensional models are presented. This section basically provides some insight into other model simulations. In Section 4 the simulation of the PR-Mesh on LARPBS is presented. In Section 5 the results of the simulation and possible future research work are outlined.

2 MODEL DESCRIPTIONS

The development of optoelectronic technologies have resulted in an increase in the usage of optical interconnects and thus optical computing. A pipelined optical bus utilizes optical fibers to transmit information. Since the propagation is unidirectional and delays are predictable, concurrent or parallel access to the optical bus is feasible thus giving rise to many models like Linear Array with a Reconfigurable Pipelined Bus System (LARPBS) [2], Linear Pipelined Bus (LPB) [4], and Linear Array with Reconfigurable Optical Buses (LAROB) [15], etc. which take advantage of the above mentioned properties. As mentioned in the earlier section the goal here is to be able to simulate an M x M processor PR-Mesh on an N processor LARPBS where $N = M$ x M. Hence in the following sections the LARPBS and PR-Mesh architectures are discussed in detail. The diagrams of all the models referenced in this section are included in the appendix.

2.1 *LARPBS Model*

The *Linear Array with a Reconfigurable Pipelined Bus System* (LARPBS) [2] consists of three waveguides. It is a one dimensional parallel processing optical model [19] [20] [21]. It can be thought of as an array, in which there are N processors P_1, P_2 ….. P_N, linearly arranged and connected by an optical pipelined bus which makes a U–turn around the processors. The processor closest to the U-turn is the head of the bus and processor farthest away from it is the tail of the bus. The bus connecting the processor is assumed to have the same length of fiber between successive processors. This implies that the 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 time steps, where a single time step comprises one bus cycle and one local computation.

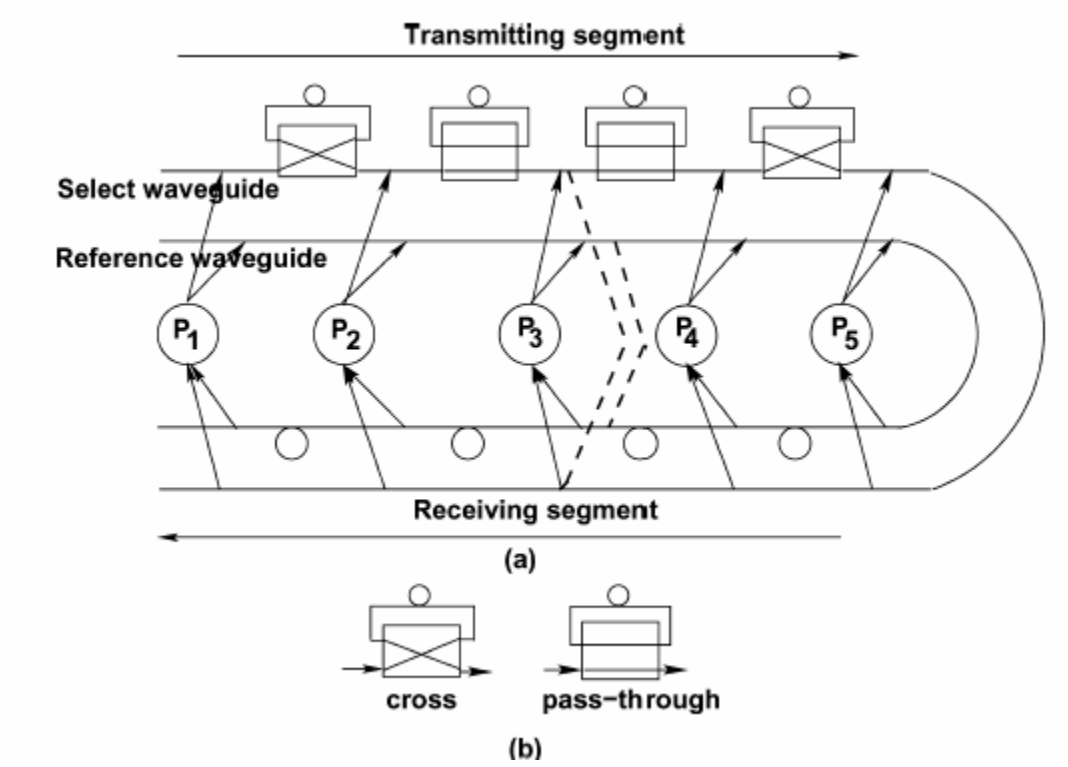


Figure 1 : LARPBS Model (a) Architecture (b) Switch Connections [12]

The LARPBS model [12] is as depicted in Figure 1. 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 data waveguide is similar to the reference waveguide and hence it is not shown on Figure 1. The bus is a U-shaped structure. The top part is used for transmission and bottom for reception. All processors are connected 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 is used to introduce a fixed propagation delay of unit time in these two waveguides. In addition, the select bus has switch-controlled conditional delays. This is added between every pair of consecutive processors P_{i-1} and P_i on the transmitting segment of the waveguide and controlled by processor P_i. The switch can function in two

positions as shown in Figure 1(b)**.** If set to cross, a unit time delay is introduced. On the other hand, if the switch is open the messages can pass-through without any delay.

2.1.1 Addressing

Though there are many addressing methods [1], the coincident pulse technique is the most common and flexible way of communication. 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 only at the desired receiver. If they coincide a double height pulse indicates to the processor to read the corresponding data frame. The coincident pulse technique uses frames for writing and addressing information. Each processor possesses a select and reference frame that has N slots for the N processors present on the LARPBS. Assuming a processor P_i wants to sends a message to another processor P_j, which is termed as the selected destination, the processor P_i transmits a message frame on the message waveguide. It selects the slot corresponding to the processor P_j on the select frame and the N^{th} slot on the reference frame. As these two frames move through the transmitting and receiving segments they will coincide at the selected destination and the processor knows that it needs to read the data frame.

In order to perform multicasting, each processor on LARPBS uses the select frame of N slots to inject a pulse into a subset of N slots within a single bus cycle. And then it chooses the rightmost slot on the reference pulses on reference waveguides. Now instead of coinciding at a single processor the pulses coincide at the subset of N processors that were selected. To broadcast messages the LARPBS injects a pulse into all the N slots of its select frame.

It must be noted that the message can be read by some other processor due to delay switches that are set. Such processors are termed actual destinations. When more than one

message arrives at a processor in the same bus cycle, it accepts only the first message and disregards subsequent messages that have coinciding pulses at the processor.

2.1.2 Reconfigurability

The strength of this model lies in the fact that it supports dynamic reconfiguration facility on the bus. There is a separate set of optical switches that exists in each waveguide of the bus. It should be noted that these are present on both the transmitting and the receiving sides. If the switches at processor P_i are set, the bus is split into two separate buses, one connecting processors P_1, P_2, . . . , P_i and the other connecting processors P_{i+1}, P_{i+2}, . . , P_n. Thus the whole model is split into two separate LARPBS structures that can work independently. The bus system can be reconfigured to allow as many separate subsystems to accommodate any need for computation and communication purposes. Further details of the model can be referred to in the paper by Pan and Li [2] [17] [18].

2.1.3 Data Movement Operations

In this section, basic algorithms designed for the LARPBS [2] that will be used in the simulation are discussed:

1. ***Broadcasting***

   In order to broadcast data across the array all conditional switches must be set to straight. A processor that wants to broadcast injects a pulse into the N^{th} slot of the reference frame and pulses in all the slots of the select frame and sends it across the respective waveguides. Thus both pulses will coincide at every processor on the bus. And all processors detect a double-height pulse and thus read the message. The broadcasting operation can take place in $O(1)$ time step [1].

2. Multicasting

While broadcasting is one to all communication operation, multicast is a one-to-many communication operation. Each processor receives only one message from a processor that wants to send a message during a bus cycle. To perform multicasting all delays switches are set straight. A processor on LARPBS that wants to send the message uses the select frame of N slots to inject a pulse into a subset of N slots within a single bus cycle. And then it chooses the rightmost slot on the reference pulses on reference waveguides. Now instead of coinciding at a single processor the pulses coincide at the subset of N *p*rocessors that were selected. The multicasting operation can take place in $O(1)$ time step [1] .

3. Binary Prefix Sum Computation

Assuming there are N binary bits held by N processors on the LARPBS say $V_0, V_1 \ldots V_{N-1}$. The aim of the binary prefix sum operation is to compute $psum_i = V_0 + V_1 \ldots + V_{N-1}$ for all $0 \leq i < N$. The binary prefix sum uses the conditional delay switches and segment switches. The binary prefix sum can be done in $O(1)$ bus cycle on an LARPBS. Processor i, for all $0 \leq i \leq N - 1$, sets its switch on the transmitting segment to cross if they hold a flag say $a_i = 1$, and straight if $a_i = 0$. All processors try and address processor $N - 1$ which is the head of the bus. Suppose that all processors to the right of processor i contains 0, all switches to the right of processor i on the select waveguide are set to straight, and thus no delay is introduced. As a result, the two pulses from processor i will coincide at processor $N - 1$. Suppose that only one processor to the right of processor i contains a 1, only one switch to the right of processor i on the select waveguide is set to cross, and thus only one unit delay is introduced. As a result, the two pulses from processor i will coincide at processor $N - 2$ and so on. After this step processor j that received the index of processor i sends a message containing its own address back to processor i. When processor V_0

receives a message containing an address *j*, it first calculates the sum of all binary numbers in the array $y = V_0 + (N - 1 - j)$, and broadcasts it to all processors on the bus. Processor *i* then gets its partial sum $Vo + V_1 + V_2 + ... + V_i = y - (N - 1 - j)$. Thus, the binary prefix sum on the LARPBS can be thus be computed in $O(l)$ time [2].

4. Compression

The compression algorithm takes a set of processors that are marked and compress them into contiguous order. These processors can be scattered across the LARPBS and may not have any particular order of occurrence. After the compression operation is performed the relative order of the occurrence of these processors is preserved. For the LARPBS this operation is performed by computing the prefix sums processors. Assume an array of *N* data elements with each processor having one data element. For example, assume that among *N* processors on the LARPBS some *s* number of processors hold a marked data value say a flag value of 1 and the rest of the processors hold a value flag value of *0*. Processors that hold the marked data are referred to as active processors. The prefix sum of all processors is computed. The objective of the compression algorithm is to move these active data elements to processors $N - s - 1, N - s . . . . . , N - 1$. Thus each processor lets the processor corresponding to the prefix sum simulate itself that is the prefix sum becomes the index of the processors that the marked elements are to be routed to. Thus the compression algorithm moves all active processors to the left side of the array [2].

2.2 *PR-Mesh Model*

The *Pipelined Reconfigurable Mesh* (PR-Mesh) [5] is a multi dimensional version of the LARPBS. The PR-Mesh can be thought of a *k*-dimensional mesh of processors, where each processor in the mesh has *2k* ports. Processors can locally manipulate their ports so as to connect to at most one other port, to form linear buses. Similar to the LARPBS the PR-Mesh also

assumes that the optically pipelined bus has the same length of fiber between successive processors, thus the propagation delays between adjacent processors are the same. The bus cycle again is considered to be the end-to-end propagation delay on the bus. The PR-Mesh can appear as a directional network since both the transmitting and receiving segments are directional. These are represented by the *T* and *R* on the figure. The architecture [5] of a PR-Mesh is as shown in Figure 2.

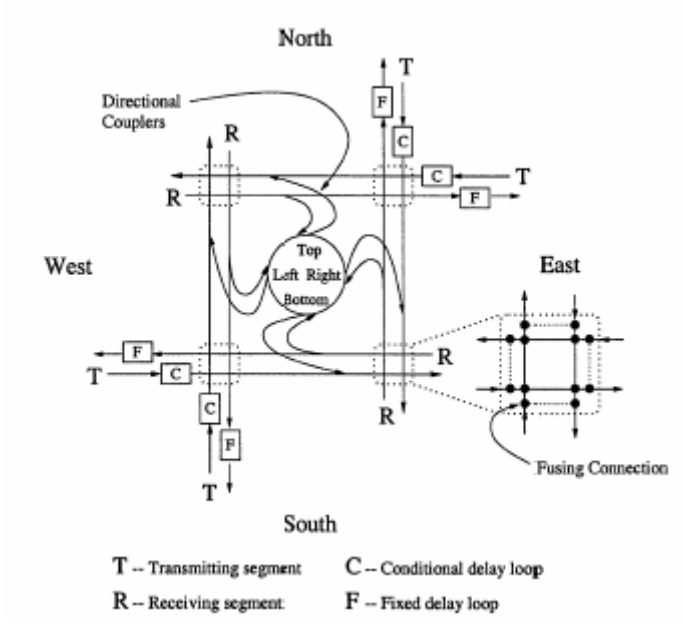


Figure 2 : PR-Mesh Processor Connections [5]

The figure represents a two dimensional PR-Mesh in which each processors has four ports. The PR-Mesh consists of an *R* x *C* [where *R* means row and *C* means column] mesh of processors, in which the four ports of the processors are joined to eight bus segments using the directional couplers. The functionality of the delay loops is similar to that of the LARPBS. There are waveguides for both the dimensions as well as directions. Similar to the LARPBS the PR-Mesh processors possesses a set of switches which they control locally to fuse/open bus segments. These are represented by the dotted boxes in Figure 2**.**

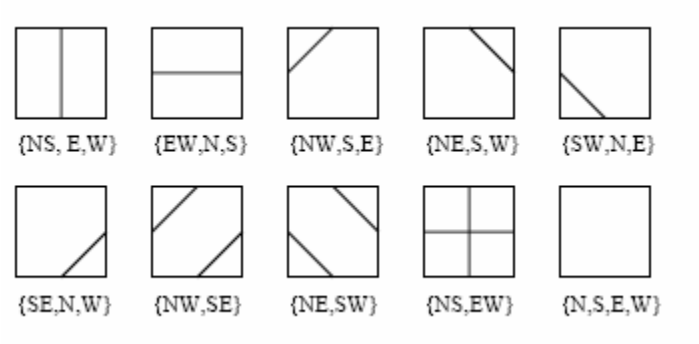


Figure 3 : Port Connections of PR-Mesh [5]

In Figure 3 the ten possible port connections for a two dimensional PR-Mesh are shown. This kind of fusing of ports in successive processors may give rise to buses that run through them to their neighbors thus forming a linear bus corresponding to the LARPBS. It must also be noted that the all buses that are formed are linear and no cycles are allowed.

2.2.1 Bus and Port Configurations

A detailed pictorial representation of the two dimensional PR-Mesh that shall be simulated by the LARPBS is shown in Figure 4**.** In the figure the two dimensional transmitting and receiving segments in both directions are represented as straight lines and the *T/R* symbol 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. The switches used for bus interconnections are divided into four quadrants as shown in dotted rectangle boxes. Each quadrant has twelve switches. The black switches are labeled *E1, E2, W1, W2* are helpful in forming row buses and switches *N1, N2, S1,* and *S2* are helpful in forming column buses, for each of the quadrants. The blue switches labeled *F1, F2, F3, F4* for each quadrant are helpful in 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. A more detailed explanation of the roles of processors with respect to buses is given in detail in further course.

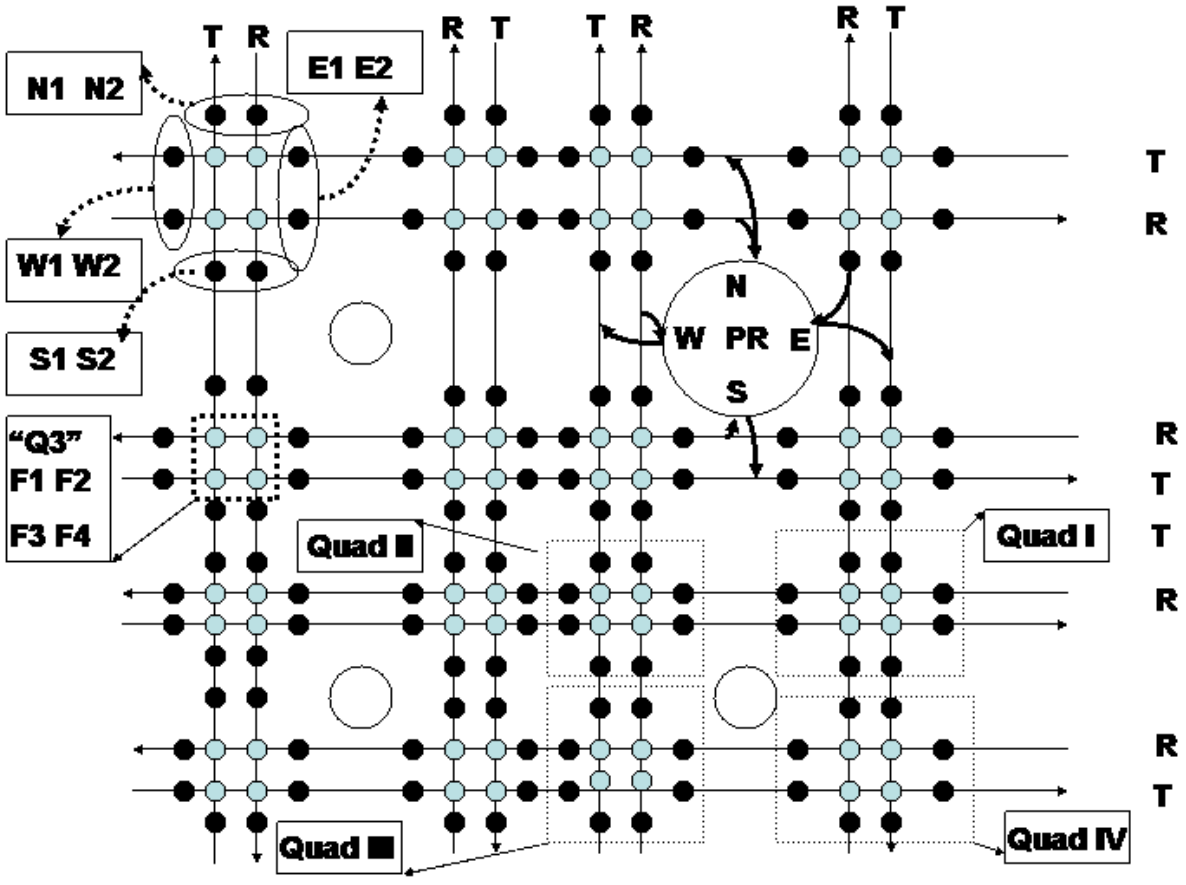


Figure 4 : Detailed view of PR-Mesh Ports & Switch Connections

2.2.2 Processors on PR-Mesh

The processors of the PR-Mesh can perform many roles as compared to the LARPBS processors. Processors of an LARPBS can be a disconnected processor, a head (processor closest to the *U*-turn), or a tail (processor farthest from the *U*-turn) or simply an intermediate processor (processors that are neither head nor tail) on the row bus. The possible roles performed by processors on the PR-Mesh are as depicted in Figure 5. The processors shown in Figure 5 correspond to the bus structure shown in Figure 6.

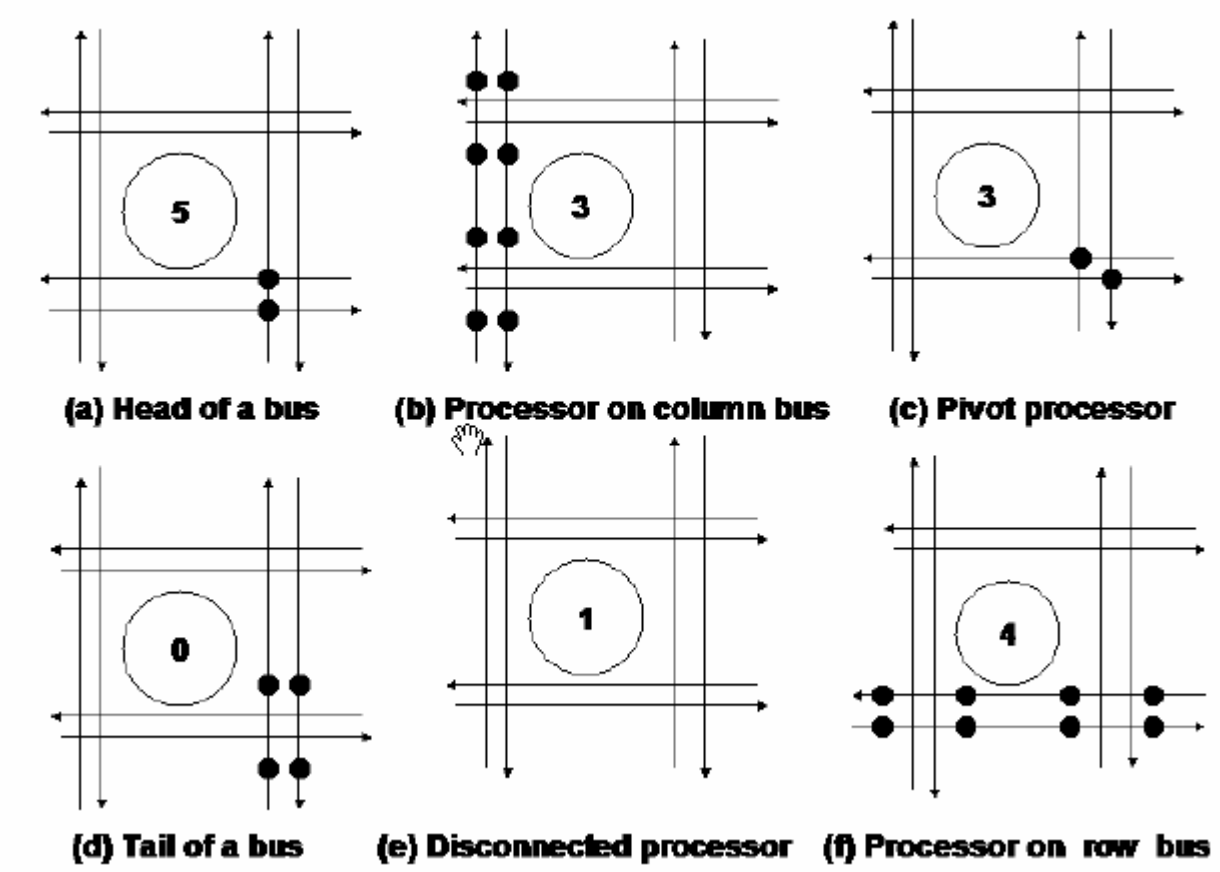


Figure 5 : Possible Roles of Processors on PR-Mesh

A PR-Mesh processor can be the head of the segment for up to two out of the four 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. An example of the head of the processor is as shown in Figure 5(a). Processors on a column bus fuse their *N1, N2, S1* and *S2* ports as shown in Figure 5(b). 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 as shown in Figure 5(d)

thereby stopping the progress of the bus along the same direction. Processors on a row bus fuse their *E1, E2, W1* and *W2* ports as shown in Figure 5(f) making the bus progress in the same horizontal direction. Processors that are not connected to any bus open all their switches as shown in Figure 5(e). Since the PR-Mesh is a two dimensional model the buses may run in both horizontal and vertical directions. It is pertinent to assume that the buses will not be a linear structure. The capability of the bus to change directions from horizontal to vertical direction or vice versa is what is called a bend. To achieve a bend the diagonally opposite ports in the same quadrant are fused as shown in Figure 5(c). The processor at which the bus bends is called a "*pivot-processor*" and it connects a horizontal and a vertical segment together.

There are eight possible ways in which a bus can bend. A bus can start transmission from east to west and bend in north or south direction. Similarly the transmission can start from west to east and bend to transmit along north or south direction. This way the bus bends from the horizontal to vertical direction. Similarly the bus can transmit from north to south and bend with in east or west direction or start transmitting from south to north and bends towards east or west direction. In this manner the bus bends from vertical to horizontal direction.

In order to understand the working of the PR-Mesh and the bus connections of processors, a small example is described. Consider a two dimensional 3 x 2 Mesh of processors with a bus structure as shown in Figure 6(a). P_5 being the head of the bus fuses the ports *F1* and *F3* in the first quadrant and P_5 and P_4 lie on the row bus thus connecting the *E1, E2 ,W1* and *W2* ports in the third and fourth quadrants. At the pivot P_3, the bus bends from a row bus to a column bus. The switches *F2* and *F4* are fused in the fourth quadrant. P_3 and P_0 lie on the column bus and they fuse their *N1, N2, S1* and *S2* ports. P_1 and P_2 are disconnected and hence open all their switches.

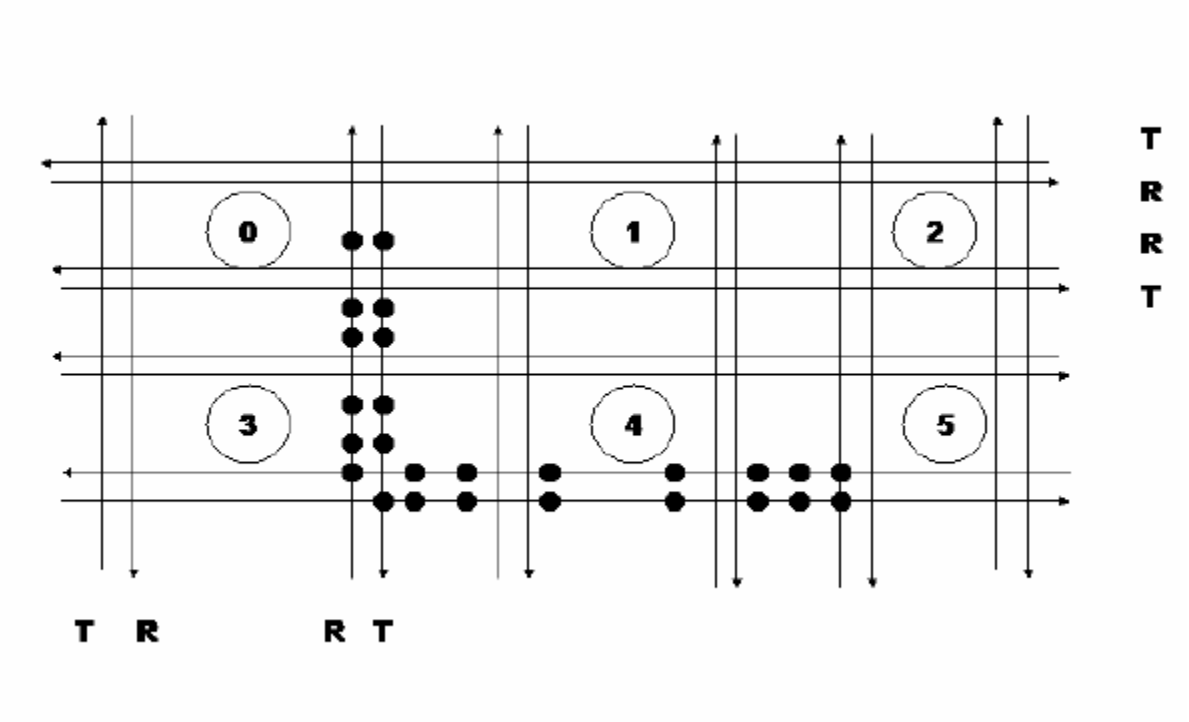


Figure 6: Bus Structure of a PR-Mesh (a) High Level View (b) Detailed View

In further sections it is assumed that the readers are familiar with the switch and port connections and different roles of processors.

3 BACKGROUND

In the introduction section the importance of studying the relationships between models was briefly mentioned. In this section this idea is further explored. In simpler terms, establishing a relationship means showing if a particular model is equivalent, more powerful or less powerful when compared to other model being compared. If a model *A* can simulate any step of another model *B* in the same amount of time it took for model *B* using same number of processors, and vice versa, then they both are considered equivalent. There has been a lot of work done in this regard. By establishing the relationship capabilities and limitations of the models can be understood. Further the presence of absence of certain features may make a model powerful than the other model to which it is being compared to. Establishing relationships helps in translation of algorithms which in turn enriches the pool of algorithms. In this section the relationships between one dimensional model and two dimensional models is discussed briefly.

3.1 *Relating One Dimensional Models*

Previous work that relates one dimensional models establishes the equivalence between the Linear Array with a Reconfigurable Pipelined Bus System (LARPBS) [2], the Pipelined Optical Bus (POB) [3] and the Linear Pipelined Bus (LPB) [4] using a cycle of simulations. For the figures of the POB and the LPB refer to the appendix. Segmentation switches that allow the LARPBS to be portioned into separate sub arrays is absent in the other two models. Also, the POB lacks the fixed delay switches. It has been proved using a cycle of simulations that the LARPBS, LPB, and POB are equivalent [11] in the sense that any algorithm proposed for one of these models can be implemented on any of the others with the same number of processors and in constant time.

The segmentation property simplifies algorithm design, but it has been proven that this functionality alone does not make the LARPBS any more powerful than the other two models that do not posses this capability. The LPB is identical to the LARPBS except that it does not posses any segment switches. The POB is also a similar model in which the conditional delay switches are positioned on the receiving side of the reference and data waveguides, rather than on the transmitting side of the select line and it also does not posses any segmentation capability.

In the simulation of the models it becomes clear that the segmentation capability must be replaced by some other property that is common to all three models. The simulation entitles usage of prefix sum algorithm that plays a key role. It efficiently utilizes the multicasting ability of the models, rather than the segmenting ability of the LARPBS. The cycle of simulations as mentioned earlier is used to establish the equivalence between the three models, i.e. initially an LPB is used to simulate an LARPBS then POB is made to simulate the LPB and finally an LARPBS simulates the POB. Each step of the simulation consists of the following three steps, namely, determining the parameters for the actual destinations of all messages, creating the select frames, and finally sending the messages.

Due to the fact that the setting of the delay switches may cause the messages to be delivered to other processors rather than the selected destination, the first step is to adjust the select frame so that the messages are sent to the selected processors. In the first step of the simulation, each processor of the LPB referred as L_i identifies if any segment switches are set in the LARPBS model that it is simulating. It has to check for any such switch because that effectively means that the LAPRBS has been divided into two separate sub-arrays. On finding such a set switch to its left, a processor L_i must inform processors ahead of it the beginning of the new sub-array. Hence it multicasts $i+1$ to each L_j (where $i<j<N$), and L_j stores this as $left_j$.

Processors that did not receive a message will assume the lowest indexed processor within its sub array to be L_0. Similarly the processors should be able to determine the sub array to it right. L_j determines the analogous position, $right_j$, by reversing the order of the processors and proceeding as before.

In the next step each processor has to determine the number of set conditional delay switches to the right of processor in its sub array which causes messages to be delivered to other processors than chosen processors. For this, the prefix sum computation is done on the set switches to the right of the processors.Now each processor has to modify the select frame so that it reflects the changes caused by the presence of delay switches and segment switches. The change is based on the information collected in the previous steps, i.e. the actual destination is found using the expression $right_j$-N+1-$psum_j$. While $psum_j$ reflects the changes due to the presence of delay switches, $right_j$ accounts for segmenting and $left_j$ is used to mask off processors that are not present in the same subarray. To send messages, all processors set all delay switches to straight and transmit their messages. If the correct processors receive the messages intended for them then the simulation is assumed successful.

The simulation of an LPB on a POB at first seems easy since both of these models lack the segmentation capability. The problems arise however, from the fact that the architecture of both models is very different. It is due to differences in the location of delay switches, the way in which it works and finally in the methods of multicasting. Here to determine the actual destination of all messages, the POB first determines the number of conditional delay switches set using the same binary prefix sums algorithm. Depending on these values, each processor of the POB can manipulate its select frame and then send the messages in the normal state of operation. The normal state of operation for an LPB is where the conditional delays are set

straight, for an LARPBS the conditional delays and segment switches are set straight and for a POB the conditional delays are set to cross.

For an LARPBS to simulate a POB, the problem lies in the fact that one select pulse in the LARPBS can address only one processor, while the POB can address multiple processors with one select pulse by setting successive conditional delay switches to straight. The processors on the LARPBS are defined as R_i and processors on the POB as B_i. In order to be able to send messages to multiple processors, the LARPBS sends messages to intermediate destinations. First, the LARPBS sends its messages to the selected destinations in the normal state of operation i.e. without modifying the select frames. Processor R_i next requests a copy of the message received by R_k in the previous step. It is quite possible that many processors might send such a request to R_k. The priority is given to the rightmost requesting processor. Then each processor R_i sets its segment switch to cross if the processor on the POB has its delay switch set to cross. The head of each subarray now broadcasts the data it received in the previous step, to forward the message to other such actual destinations, and each processor R_i now has the same message as B_i would have in the POB. Hence it has been proved that the LARPBS, LPB, and POB are equivalent models. Each one can simulate any step of one of the other models in $O(1)$ steps with the same number of processors.

3.2 *Relating Two Dimensional Models*

In the previous section equivalence was proved by an automatic mapping of algorithms with respect to their functionalities without any loss of speed or efficiency among the models. In this section the issues in relating two dimensional models is discussed briefly. The main problem associated with two dimensional models is the number of configurations possible due to the multiple dimensions. Keeping this factor in mind, their equivalence is denoted in a slightly

different manner. Here the complexity is measured by relating their time to within a constant factor and the number of processors to within a polynomial factor. Mentioning some of the major unifying methods it has been established that the PR-Mesh has the same complexity as the cycle-free Linear Reconfigurable Network (LR-Mesh) [13]. In the paper it is proved that in constant time, using a polynomial number of processors, the cycle-free LR-Mesh can solve the same class of problems as the LR-Mesh. It can be inferred that the PR-Mesh can solve the same class of problems within the same order of steps using polynomial processors. The complexity class is then extended to accommodate two other optical models, namely the Array with Reconfigurable Optical Buses (AROB) [6] and the Array Processors with Pipelined Buses using Switches (APPBS) [8]. The AROB is a two dimensional expansion of the LAROB. The main features of the AROB include an internal timing circuit that is capable of counting *petit-cycles*. Among other functionalities the AROB is capable of bit polling, capacity to shift the select frame with accordance with the reference pulse by adding up to *N* unit delays and an enhanced model that is capable of changing switch settings during bus-cycles. Similar to the AROB an APPBS also allows the processors to change their switch configuration in midst of a bus–cycle and within *petit- cycles.* [It denotes the delay between a processor and its adjacent neighbor.]

Though the PR-Mesh does not possesses any of these functionalities of the AROB and the APPBS it has been proved that with a polynomial increase in the number of processors the PR-Mesh is capable of simulating both the AROB and the APPBS. For detailed explanation of the simulation please refer to Trahan *et al.* [1]. For figures please refer the Appendix. These were some of the major accomplishments in unifying the reconfigurable optical models and relate them to other more widely known models [14].

4 SIMULATING A PR-MESH MODEL ON AN LARPBS

Before the simulation of the PR-Mesh on a LARPBS can be begun it is necessary to know how the processors on a PR-Mesh are to be mapped to a LARPBS model. This basically identifies how the processors simulating the PR-Mesh processors are to be placed in the LARPBS. So for this, a simple row-major arrangement of processors on PR-Mesh based on their index is sufficient for the initial linear arrangement the processors in the LARPBS.

Some of the aims of the simulation will be:

1. I***dentification and Ranking of Components:*** Components here refer to the number of independent buses that can be present on the PR-Mesh. Here simulation is performed for a two dimensional PR-Mesh and hence buses can be on *x*-axis alone or *y*-axis alone or can be on both the axis. Buses panning across both the dimensions must be treated as a single bus. The components must be ranked and at the end of simulation each of the buses must be represented as a separate sub-array in the LARPBS and arranged in the descending order of rank. It has to be noted that processors that do not belong to any bus must be treated as thought they are the only component of a bus.
2. ***Identification of Component-Members:*** Component members refer to the processors that are connected to each of the buses. Since it is known that each processor on the PR-Mesh can be connected to multiple buses when simulating on the LARPBS it must be determined which processors belong to which components or bus segment. A detailed explanation of situations when processors are connected to multiple buses is explained in further course of the simulation. Processors that are not connected to any bus can be ranked separately after all the processors that belong to a bus or they can appear in between two sub-arrays.

3. ***Ranking Component-Members:*** The next step in the simulation is the ranking of the processors that belong to a particular bus. Referring to Figure 6**,** it must be noted that P_5 is ranked 0 since it is the head of the bus. While P_4, P_3 and P_0 are ranked 1, 2 and 3 respectively. It is possible that the PR-Mesh may form cycles while configuring the bus as depicted in Figure 7.

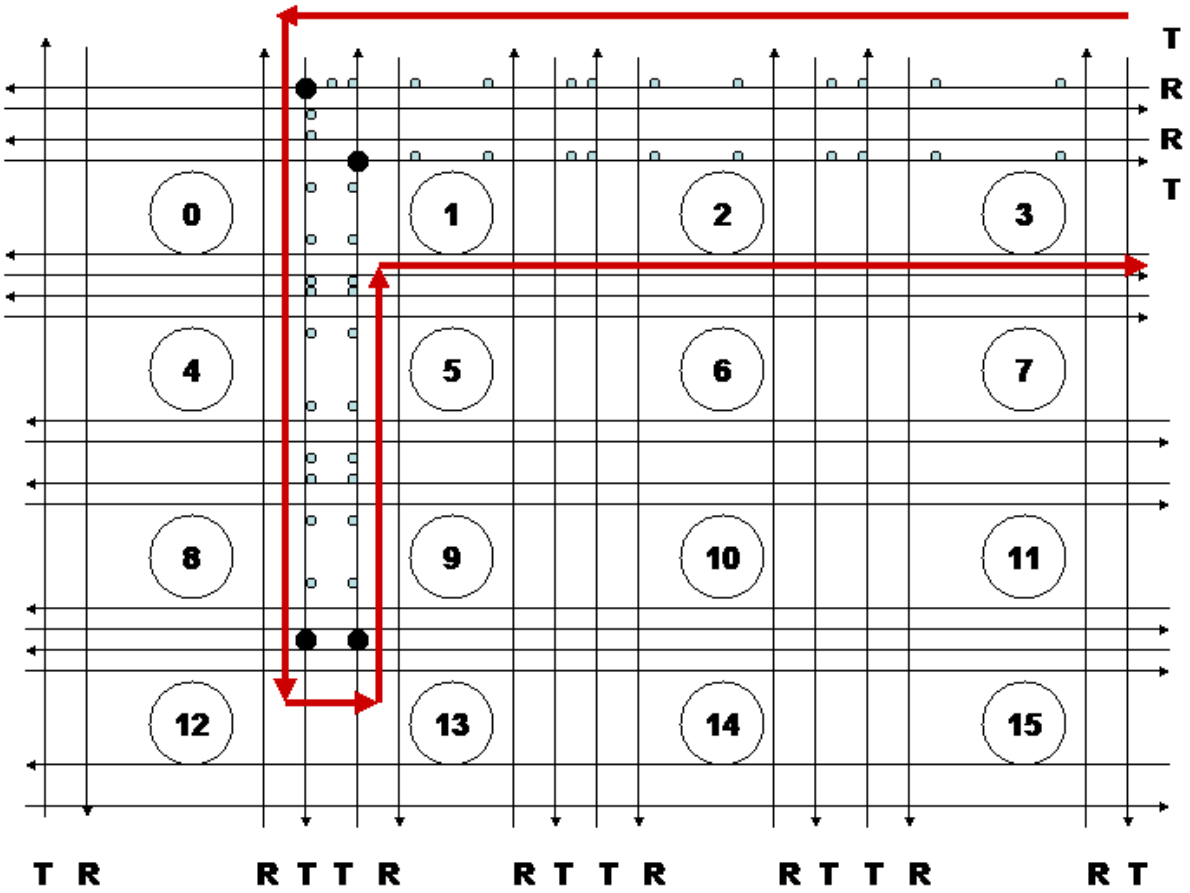


Figure 7 : PR- Mesh permitting Cycling

It can be seen from the Figure 7 that the PR-Mesh can form cycles as processors P_1, P_2 and P_3 are on the bus twice as shown by the darker line. The dots represent the change in directionality of the bus. But the processors in the loop can only receive messages only once

during a bus cycle hence rendering cycling of buses useless. Therefore it is legitimate to assume that a processor cannot appear twice on a bus, i.e. there can be no cycles.

4. ***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.

In the forthcoming sections the simulation is performed by considering simpler to more complex bus configuration patterns. The first case (CASE 1-a) deals with the assumption that the processors are connected to at most one bus and the bus is bent at most once. This case is further refined in the subsequent section (CASE 1-b) since it fails to effectively segment the row and column segments when the ID of the head of the bus is lower than that of the rest of the processors. The next case (CASE 1-c) assumes the processors are connected to at most one bus and the bus can be bent any number of times. The maximum possible bends that the bus can have, which also represents the worst case scenario is also discussed. The next section (CASE 2-a) assumes the processors to be present across multiple buses while the bus is allowed to be bent only once and the final case (CASE 2-b) deals with processors across multiple buses and buses can have multiple bends.

4.1 *Simulating a PR-Mesh model on an LARPBS – Case 1(a)*

This case involves simulating an *M x M* PR-Mesh using *N* processor LARPBS. The assumptions made here are that the processors on the PR-Mesh 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 the buses using the same number of processors.

Overview:

This section describes the high level operations needed to be performed for this simulation.

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

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 - Case 1(a)

Simulation:

The following section describes the actual simulation process with details about each step described in the overview.

Model: An N processor LARPBS, where $N = M \times M$.

Input: An M x M PR-Mesh

Output: Processors in a bus grouped together in the order in which they lie on PR-Mesh

Assumptions: For simulating the PR-Mesh on the LARPBS the following assumptions are made

1. Each Processor on PR-Mesh is connected to at most one bus.
2. Each bus has just one bend.

Steps:

Begin

1. Processors which are the head of a segment and processors that are completely disconnected; set flag as 1.
2. Compress all processors holding flag value as 1.

3. Compute the prefix sum of each of these processors; this denotes the ranks of each processor (B_{RANK}).
4. Arrange the processors in row major order
5. Forming row segments
    a. All processors which are pivot nodes (where their index *i*; *i* Mod *M* is not equal to 0), completely disconnected, head of segments and processors whose East-West port are not connected set their segment switches.
    b. Processors whose West Port is not connected send their index (N_R) to the head of the segment. Disconnected processors take their corresponding indices as $N_{R.}$
    c. If the processor that sent message in step 5b is a head of the bus send a value of 1 as (H_{SEG}) to the head of the segment and also the B_{RANK}. Disconnected processors assume value of H_{SEG} as 1.
    d. Processors that received message in step 5b send its index as ($P_{PIVOT\text{-}NODE}$) as well as received index and finally the value of H_{SEG} to all processors between them and the end node (including the end node).
    e. All nodes that received message in previous step set flag as 1.
    f. Compute the prefix sum of all these processors (P_{RANK}). [Subtract 1 to start Ranking from 0].
    g. Processor with index $P_{PIVOT\text{-}NODE\ -\ 1}$ sends its prefix sum ($N_{Row\text{-}Sum}$) to the pivot node. [Must add one to $N_{Row\text{-}Sum}$ as ranks begin from 0]
6. Arrange Processors in column major order.
7. Forming column segments

a. All processors whose North port is not connected or completely disconnected set their segment switches.

b. All pivot nodes and processors whose North Port is not connected send their index (N_C) to the head of the segment. Also they send the $N_{Row\text{-}Sum}$ which is stored as P_{TOTAL}. Disconnected processors take their corresponding indices as N_C.

c. If the processor that sent message in step 6b is a head of the bus send a value of 1 as (H_{SEG}) to the head of the segment and also the B_{RANK}. Disconnected processors assume value of H_{SEG} as 1.

d. Processors that received a message in step 6b send its index as (P_{HEAD}) as well as received index and finally the value of H_{SEG} as 1 if it is the head of segment to all processors between them and the end node (including the end node).

e. All processors now compute their ranks (P_{RANK}). [Subtract 1 to start Ranking from 0].

f. Processor with index N_C sends the rank ($N_{Col\text{-}Sum}$) to the node whose index it received in Step 6b.

8. Re-Ranking Processors on the bus

a. If pivot node received H_{SEG} =1 in Step 6

i. The processors along the column segment retain their P_{RANK}.

ii. Form row segments and pivot node broadcast $N_{Col\text{-}Sum}$ and $N_{Row\text{-}Sum}$ to all processors in row segment.

iii. All processors adjust their ranks as $N_{Col\text{-}Sum} + ((N_{Row\text{-}Sum} - 1) - P_{RANK})$.

iv. Pivot node sends total number of processor to the head to the bus (P_{TOTAL}).

b. If pivot node received H_{SEG} =1 in Step 5

i. Pivot node adds 1 to $N_{Row\text{-}Sum}$ and broadcasts to all processors along column segment.

ii. All processors adjust their ranks as $N_{Row\text{-}Sum} + P_{RANK}$.

iii. Pivot node sends total number of processor to the head to the bus (P_{TOTAL}).

9. All disconnected and head of segments holds the value of P_{TOTAL} and a prefix sum ($N_{SLOT\text{-}START}$) is performed on computed for each segment.

10. Form Row and column segments again and broadcast the $N_{SLOT\text{-}START}$ value.

11. Each processor computes its new index as $N_{SLOT\text{-}START} + P_{RANK.}$

12. Each processor then arranges itself according to the new index.

End

4.1.1 Explanation

The algorithm in the preceding section is explained with the help of an example shown below.

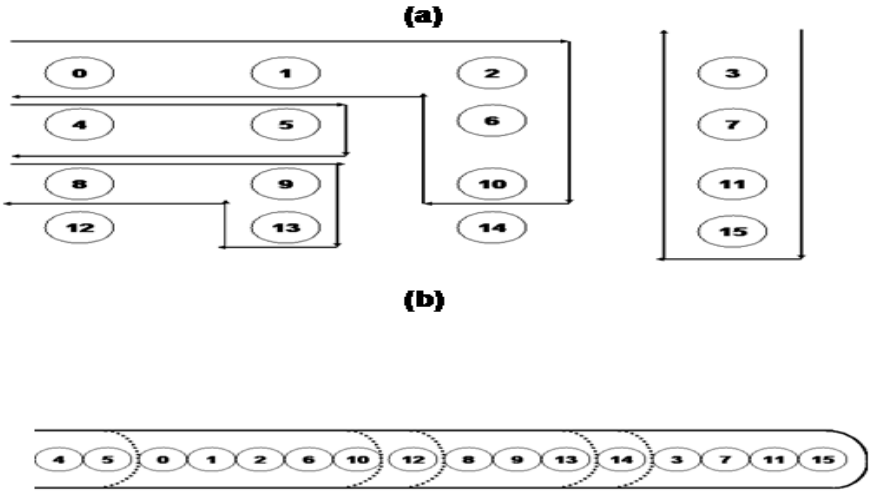


Figure 8 : Mapping Processors (a) Processors on PR-Mesh (b) Processors on LARPBS

In the overview section the steps of the simulation is described in terms of major steps. In the simulation how this aims set in the overview section are achieved by the processors is described. In this section a further elaboration of the simulation process is described. Figure 8 (a) shows the processors on the PR-Mesh and (b) shows how the processors have to be arranged so as to simulate the PR-Mesh on LARPBS. In other words, the (a) part is the input to the LARPBS and (b) part is the output that is expected after the simulation is performed. The first step in the algorithm is to determine the number of buses that exist on the PR-Mesh. It should be understood that each disconnected processor must be assumed to lie independently on a bus. Further, the order in which these buses are present, i.e. the rank of each bus is to be determined. This detail is provided by the "B_{RANK}" variable. This value at the beginning of the simulation is held by the head of the segments and the disconnected processors which are basically thought of as the head of the segment in which these are the only processors.

In the next few stages, the processors that lie across the same bus are to be identified. Processors may lie on a along a row bus or a column bus or on a row as well as column bus as shown in Figure 8. One important thing to notice from Figure 8 is that pivot processors (that form a bend as shown in Figure 5(c)) are key in identifying the processors that lie on the same bus. It becomes clear as the algorithm progresses. In order to find processors along the same bus, the first step is to find processors that lie on the horizontal part of the bus and then to temporarily rank (P_{RANK}) them if these processors lie on the row as well as column bus. It is a temporary ranking as the rest of the processors along the bus are not known at this stage and also the direction of the head processor is not known. However, for processors that entirely lay on a horizontal bus the ranking will be permanent. Once the horizontal segments of the processors are

identified by arranging processors in row-major format, the pivot processor now knows the following facts:

1. The number of processors along a row segment ($N_{Row\text{-}Sum}$)
2. If the head of the segment lies in this segment (H_{SEG})

Now the processors that are 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. Once this step is completed the above mentioned facts hold true for this step of simulation also. Now 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 now processors along the pivot-tail of the segment need to re-adjust their ranks.

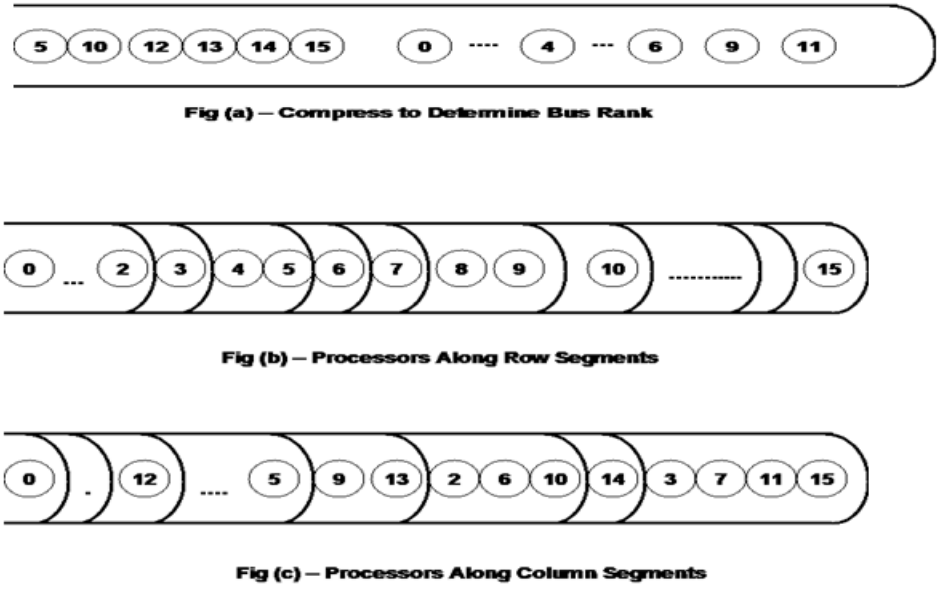


Figure 9: Simulation Algorithm Case 1(a)

The final step of the algorithm is finding which slot is to be occupied by which bus ($N_{SLOT\text{-}START}$). This is calculated by the computing the prefix sum of total number of processors along each bus as they are ranked. All the processors need to know is the beginning of the slot as they are ranked; the new index is easily calculated. The table shown below describes in detail the variables and values received by processors during the simulation.

Table 1 : Values received by Processors during the Algorithm 1(a)

		PROCESSORS ON LARPBS															
Steps	**Variables**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
1	Flag						1					1		1	1	1	1
3	B_{RANK}						0					1		2	3	4	5
5.b	N_R			0			4				8			12		14	
5.c	H_{SEG}													1		1	
5.c	B_{RANK}													2		5	
5.d	$P_{PIVOT-NODE}$ & H_{SEG} & B_{RANK}	2 & 0	2 & 0			5 & 1 & 0				9 & 0							
5.e	Flag	1	1			1				1							
5.f	P_{RANK}	0	1			0				0							
5.g	$N_{Row-Sum}$			2							1						
6.b	N_C											2			9		3
6.c	H_{SEG}			1	1						1			1		1	
6.d	P_{HEAD}			10	15			10	15		13		15				15
6.d	B_{RANK}			1	5			1	5		3		5	2	3	4	5
6.e	Flag			1	1			1	1		1		1				1
6.f	P_{RANK}			0	0			1	1		0		2		1		3
6.g	$N_{Col-Sum}$			3							2						
7.a.ii	$N_{Col-Sum}$	3	3							2							
7.a.iii	P_{RANK}	5	4							3							
8.	P_{TOTAL}						2					5		1	3	1	4
8.	$N_{SLOT-START}$						0					2		7	8	11	12
9.	B_{RANK} & $N_{SLOT-START}$ & index	1 & 2 & 2	1 & 2 & 3	1 & 2 & 4	5 & 12 & 12	0 & 0 & 0	0 & 0 & 1	1 & 2 & 5	5 & 12 & 13	3 & 8 & 8	3 & 8 & 9	1 & 2 & 6	5 & 12 & 14	2 & 7 & 7	3 & 8 & 10	4 & 11 & 11	5 & 12 & 15

4.1.2 Complexity Analysis

The following section describes in detail the complexity analysis of the algorithm discussed in this section. The steps discussed below take into account complexity of each and every step of the algorithm and give a final value based on the summation of these steps. The complexity analysis is this section as well as the others is based on the algorithms designed for the LARPBS. The sources are sited for each and every step.

The complexity of steps in the simulation algorithm is as follows:

1. Compression algorithm takes $O(1)$ time [2].
2. For processors to compute their temporary ranks along a row or column bus takes $O(1)$ time [rank = index of head of segment – index of the processor computing its rank] [2].
3. 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].
4. All communication between processors [this includes communication between two processors, multicasting or broadcasting] takes $O(1)$ time [2].
5. 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 [2].
6. The prefix sum is computed for the head of the segments. For integers with bounded magnitude algorithm for prefix sum computation, takes $O(\log \log N)$ time using N processors [12].
7. The permutation routing of the processors in LARPBS takes $O(1)$ time [11].

It is to be noted that the efficiency of this simulation lies in the efficiency of computing the prefix sum of integers with bounded magnitude. Hence in the future, the efficiency of the

simulation is likely to improve if the prefix sum computation algorithm can be made to work in a more efficient way.

Lemma 1: Each step of an M x M processor PR-Mesh, in which each processor is connected to at most one bus and the bus can have at most one bend, can be simulated by an N (where $N = M$ x M) processor LARPBS in O(log log N) time.

4.2 *Simulating a PR-Mesh model on an LARPBS – Case 1(b)*

From the diagrams shown in Figure 8(a) it can be seen that all the buses run in the same direction and the index of the head is always higher than the other processors along a bus. Hence on implementing the algorithm Case 1(a) for cases where the buses run in opposite directions and with a head processor with lower index than the other processors; the algorithm does not work any more as shown in Figure 10.

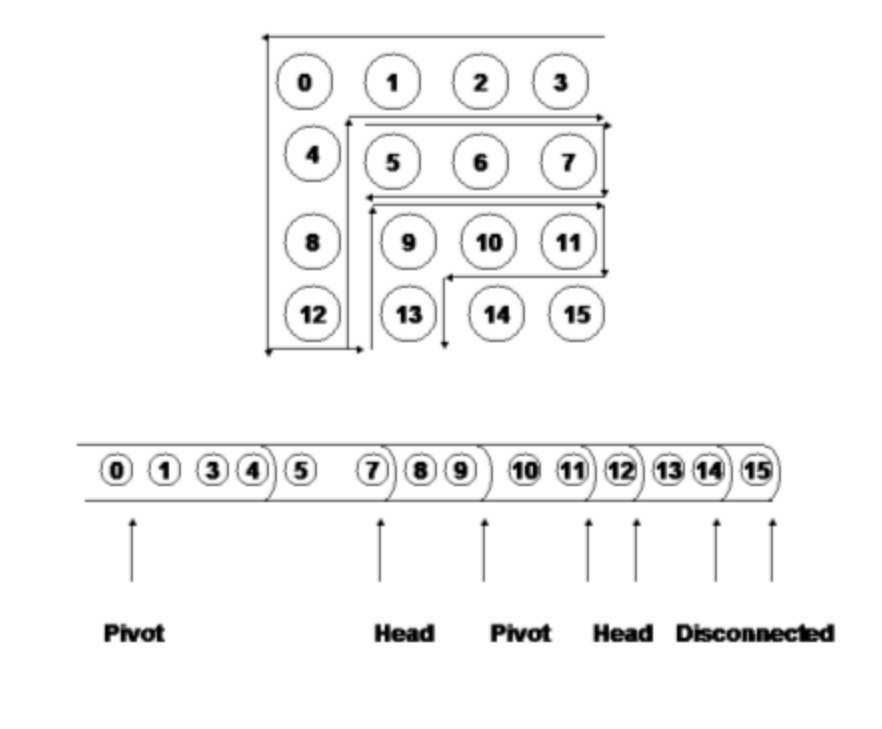


Figure 10 : Need for Refinement for case 1(a)

From the Figure 10 it can be seen that processors zero to three lie on the same row bus, but since the pivot node‘s index is lower than rest of the processors in that row bus, the segmentation causes processor P_9 alone to lie on the row bus and the other two processors are separated. Hence the algorithm had to be improved to accommodate the aforementioned conditions. It can be seen that the problem lies only in the identifying the row and column segments. Hence once this is

overcome the algorithm can function as before. The identification and ranking of the bus is similar to the previous case, Case 1(a)) Steps 1 to 3. The model, input, output and assumptions are same as the previous case. The following steps outline the procedure for identifying row/column segments.

Begin

1. Arrange the processors in row major order
2. Forming row segments
    a. All processors whose East-West ports are not connected as shown in Figure 5(f) set their segment switches.
    b. All processors who segmented in previous step send their index to its left/right neighbor processor as shown in the Figure 11(a). The messages are denoted by the arrows below the processors. [The left neighbor is the processor sending its *index* – 1, and right neighbor is processor sending its *index* +1]. All disconnected processors also do the same.
    c. If a processor receives a message, it must set its segment switch. Received processor now knows that 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.
    d. Now the algorithm can proceed as before.

End

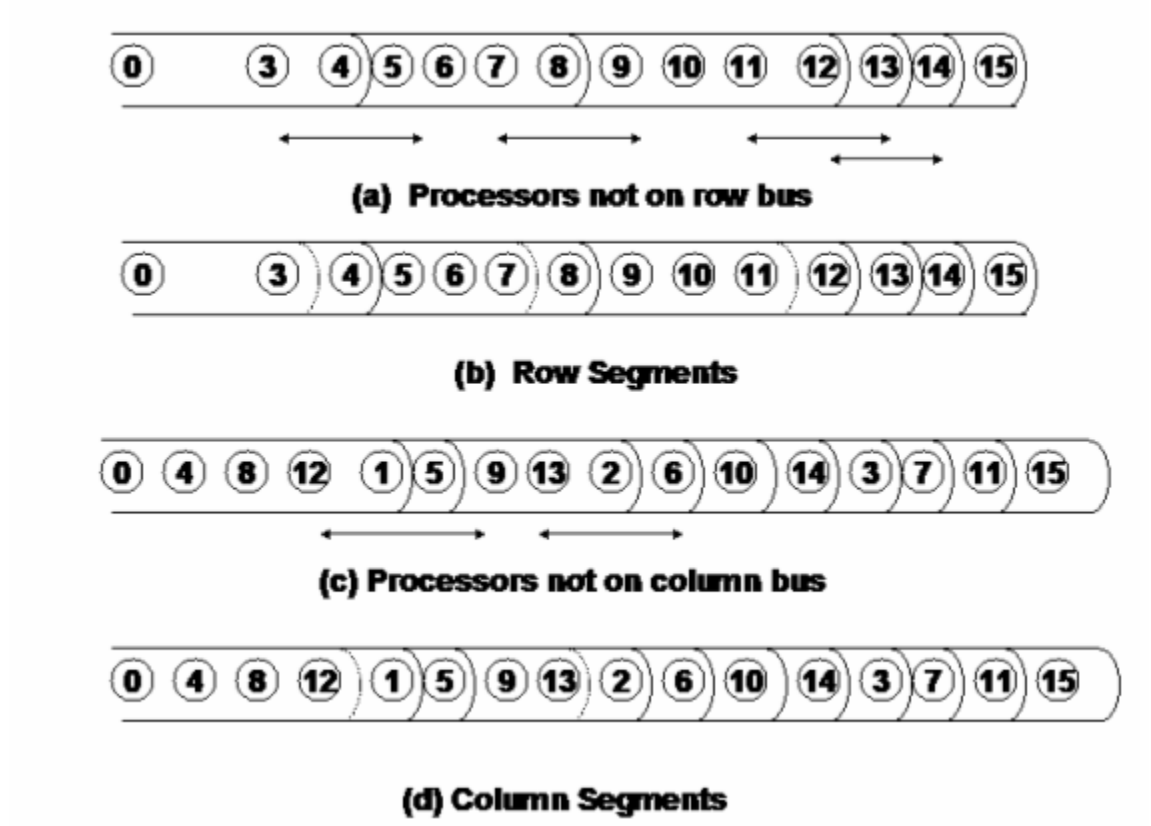


Figure 11 : Simulation Steps for case 1(b)

3. Forming column segments.
    a. All processors whose North South ports are not connected as shown in Figure 5(b) set their segment switches.
    b. All processors who segmented in previous step send their index to its left/right neighbor processor. [Left neighbor is processor sending its *index – M* for the processor above on the column and right neighbor is processor sending its *index + M* for the processor below on the column]. All disconnected processors also do the same.
    c. If a processor receives a message, it must set its segment switch. Received processor now knows that processor on its left/right had set its segment switch since it did not lie on a column bus therefore it cannot also lie on the column bus that the receiving processor lies on.

d. Now the algorithm can proceed as before.

End

4.2.1 Complexity Analysis

The only changes made to the algorithm lie in the communication between the processors and therefore bear no effect on the complexity. Thus there is no change in Lemma 1.

4.3 *Simulating a PR-Mesh model on an LARPBS – Case 1(c)*

In the simulations in the previous sections it is not realistic to neither assume that the buses in the PR-Mesh are one dimensional nor have only one bend between a horizontal and a vertical direction. It is reasonable to assume that the bus will be bent multiple times, indicating that the directionality of the bus changes many times. This makes it difficult to preserve the order in which the processors lie on the bus. This case modifies the assumption in the previous case. It is still assumed that the processors are still connected to at most one bus as shown in Figure 12.

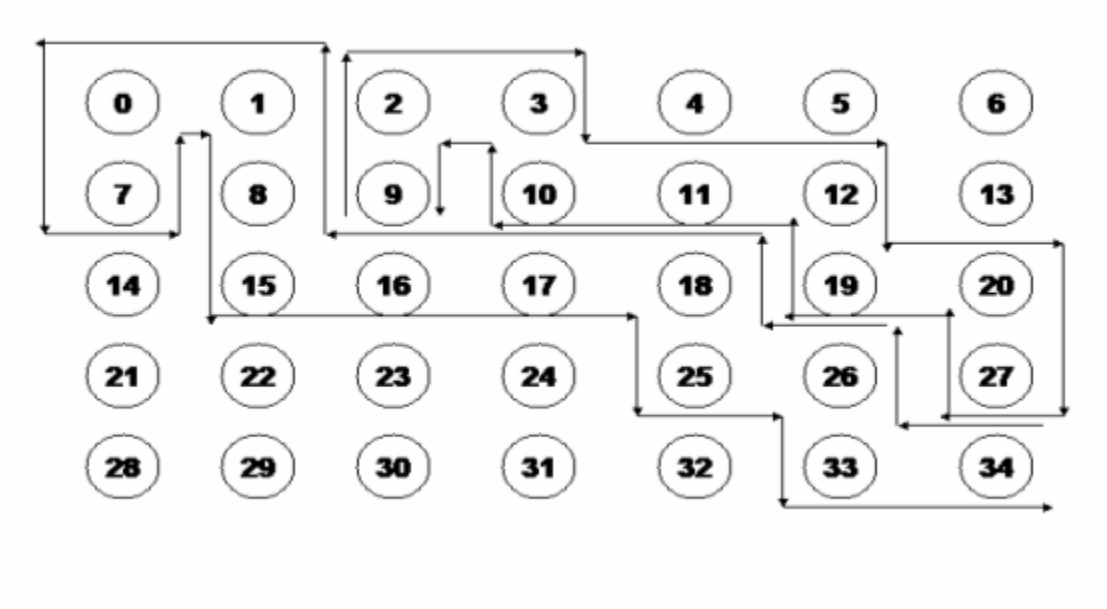


Figure 12 : Processors on PR-Mesh on Bus with Multiple Bends

Overview:

This section describes the high level operations needed to be performed for this simulation.

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

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 (denotes total processors on bus)

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 - Case 1 (c)

Simulation:

The following section describes the actual simulation process with details about each step described in the overview.

Model: An N processor LARPBS, where $N = M \times M$.

Input: An M x M PR-Mesh

Output: Processors in a bus grouped together in the order in which they lie on PR-Mesh

Assumptions: For simulating the PR-Mesh on LARPBS the following assumptions are made

1. Each Processor on PR-Mesh is connected to at most one bus.
2. Each bus can have multiple bends.

Steps:

Begin

1. Processors which are the head of a segment and processors that are completely disconnected; set flag as 1.
2. Compress all processors holding flag value as 1.

3. Compute the prefix sum of each of these processors; this denotes the ranks of each processor (B_{RANK}).
4. Arrange the processors in row major order
5. Forming row segments
    a. All processors whose East-West ports are not connected as shown in Figure 5(f) set their segment switches.
    b. All processors who segmented in previous step send their index to its left/right neighbor processor as shown in the Figure 11(a).The messages are denoted by the arrows below the processors. [The left neighbor is processor sending its *index* – 1, and the right neighbor is processor sending its *index* +1]. All disconnected processors also do the same.
    c. If a processor receives a message, it must set its segment switch. Received processor now knows that 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.
    d. The head and tail segment processors exchange their indices and also the head informs all the processors between itself and the tail the head and the tail processor index.
4. Forming column segments.
    a. All processors whose North South ports are not connected as shown in Figure 5(b) set their segment switches.
    b. All processors who segmented in previous step send their index to its left/right neighbor processor. [The left neighbor is processor sending its *index* – *M* and the

right neighbor is processor sending its *index + M*]. All disconnected processors also do the same.

c. If a processor receives a message, it must set its segment switch. The receiving processor now knows that processor on its left/right had set its segment switch since it did not lie on a column bus therefore it cannot also lie on the column bus that the receiving processor lies on.

d. The head and tail segment processors exchange their indices and also the head informs all the processors between itself and the tail the head and the tail processor index.

5. Ranking processors on the bus

   a. After forming row and column segments it can be seen that at each pivot processor, in order to rank the processors in its segment, it needs the number of processors ahead of it and hence this problem in simple terms boils down to calculating the prefix sum of the number of processors lying ahead of it. Rank processors (P_{RANK}) on the bus. Detailed working of the ranking process is provided in the explanation section.

   b. All new pivot processors learning the identity of the head of the bus must communicate with the head, to convey their IDs. This is vital since the ID is used by the head to convey the beginning slot value ($N_{SLOT\text{-}START}$) to all pivot processors.

6. All disconnected and head of segments holds the total number of processors (P_{TOTAL}) on that bus and then a prefix sum ($N_{SLOT\text{-}START}$) is computed for each bus.

7. The $N_{SLOT\text{-}START}$ value is then sent to each pivot processors so that it can be broadcasted to all processors on the bus.

8. Form Row and column segments again and broadcast the $N_{SLOT\text{-}START}$ value.

9. Each processor computes its new index as $N_{SLOT\text{-}START} + P_{RANK.}$

10. Each processor then arranges itself according to the new index.

End

4.3.1 Explanation

The initial steps of the simulation are same as the previous sections. For detailed explanation for these steps refer to Section 4.1.Here a detailed explanation of the prefix sum computation using a binary tree-like method is furnished. It can be observed 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. Here the bus that runs through processors P_7, P_0, P_1, P_8, P_{15}, P_{16}, P_{17}, P_{18}, P_{25}, P_{26}, P_{33} and P_{34} of Figure 12 is considered for explanation purposes. On first forming the row bus, both processors lying on the row buses (for example P_0 - P_1, P_{15} through P_{18}, P_{25} - P_{26} and P_{33} - P_{34}) become aware of the pivot processors that they might need to communicate with in order to know the number of processors ahead of them. At this point it has to be noted that the head of the bus is not known.

After forming the column segments, the pivot processors gain knowledge of another set of pivot processors and also the number of processors in their segment. For example, after forming column segments processor P_0 becomes aware that there is just one processor ahead of it processor P_7 and it is the head of the bus. Now the directionality of the bus is learned by a new processor and must be passed on to other processors. This can be done only by the pivot processors. And ranking the processors in their segments can be done only after learning the number of processors ahead of them. This has now become a prefix sum computation on the

number of processors held in each segment. The binary tree structure that is ideally used for the prefix sum computation for the bus is as shown.

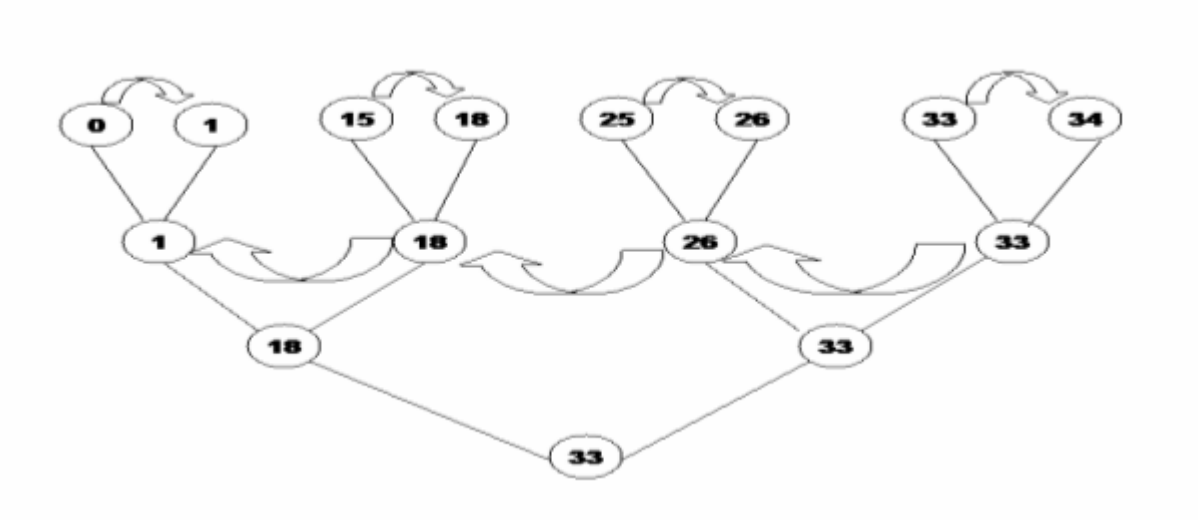


Figure 13 : Prefix Sum Computation for the Pivot Processors

The main problem why the traditional binary tree method cannot be used here can be seen from the diagram. Unlike the traditional binary tree method, the index of the processor in the next step is not known. For example if there were processors ranked P_0 to P_5 for which the prefix sum is to be computed then it is known that in first stages processors communicate in pairs like P_0- P_1 , P_2 –P_3, P_4 -P_5. In the next stage processors P_1 -P_3 know they have to communicate since the indices are increasing in a uniform manner. But here that is not the case.

Hence in order to solve that problem, row and column segments are formed and the new processor that has learnt of the directionality and the bus rank steps up and has to proceed to the next stage so as to provide information. After the first communication step P_0 learns of the identity of the head of the segment and computes its rank based on the prefix sum. Each processor computes the rank by adding the prefix sum to the number of processors held in its own segment. For example, the prefix sum of P_0 is 1 and it computes its rank by adding 1 to the

number of processors held in its segment which is 1 and subtracting 1 as rank starts with 0. So the rank of P_0 is 1+1-1 = 1. In the second stage the row segments are formed again. Now processor P_1 is the processor that knows the prefix sum of pivot processors ahead of it as it communicates with P_0 again and can becomes the processor ahead of which prefix sums are already computed. Subsequent pivot nodes have to communicate with this node to get the information on the bus.

For segments where the processor has not yet learned of the identity of the rank and ID of the head of the bus, it must be informed by processors that have this information farther in the direction of transmission. For communication purposes all the set 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. For example in the first step processor P_{15} communicates with P_{18} and informs it the id of P_1. The detailed communication between processors on the bus for prefix sum computation is as shown in Figure 14.

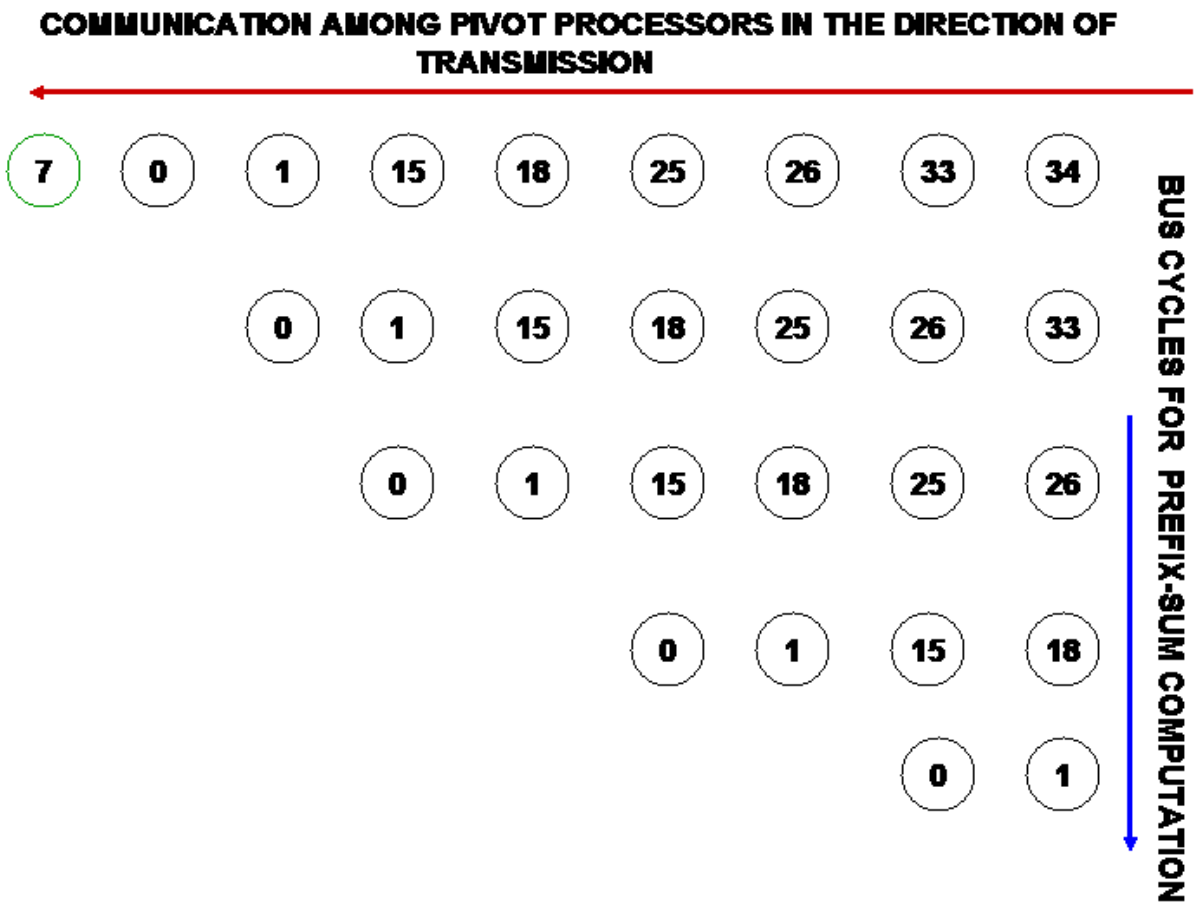


Figure 14 : Communication among Pivot Processors

Taking the example of processor P_{34} from the Figure 14 in the first stage of the row formation communicates with processor P_{33}. After the column segments are formed and processor P_{33} becomes aware of processor P_{26}. Now the prefix sum computation phase begins and it is indicated by the blue arrow mark. The arrow also is representative of the number of steps required for this operation. During the prefix sum computation phase again processor P_{34} communicates with processor P_{33}. During this step three important actions take place processor P_{34} must provide processor P_{33} its index for communication purposes. In the second step processor P_{33} sends the prefix sum it has computed so far to processor P_{34} and during the final step processor P_{33} provides the index of processor P_{26} to processor P_{34}. During the next phase

processor P_{34} communicates with processor P_{26} which provides it with the index of processor P_{18} and so on.

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 which this bus needs to occupy depending on the rank.] This is continued until the last step in which the final node posses 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 done rest of simulation is similar to the previous section.

4.3.2 Complexity Analysis

The following section describes in detail the complexity analysis of the algorithm discussed in this section. The steps discussed below take into account complexity of each and every step of the algorithm and give a final value based on the summation of these steps.

The complexity of steps in the simulation algorithm is as follows:

1. Compression algorithm takes $O(1)$ time [2].
2. For processors to compute their temporary ranks along a row or column bus takes $O(1)$ time [rank = *index of head of segment – index of the processor computing its rank*] [2].
3. 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].
4. All communication between processors [this includes communication between two processors, multicasting or broadcasting] takes $O(1)$ time [2].

5. All the internal functions that the processors perform for e.g. adjusting their ranks once the other processors along the bus, finding number of processors along their segments etc takes $O(1)$ time [2].
6. Prefix sum computation using the binary tree method takes about $O(\log b)$ time using N processors where b denotes the number of bends in the bus [1].
7. For integers with bounded magnitude algorithm for prefix sum computation takes $O(\log \log N)$ time using N processors. This is done to find the prefix sum of processors in each bus to find the slot which the next ranked processors needs to occupy [12].
8. The permutation routing of the processors in LARPBS takes $O(1)$ time [11].
9. The total time taken to run the simulation is $O(\log \log N + \log b)$ where b denotes the number of bends in the bus.

Lemma 2: Each step of an M x M processor PR-Mesh, in which each processor is connected to at most one bus and the bus can have more than one bend, can be simulated by an N (where $N = M$ x M) processor LARPBS in $O(\log \log N + \log b)$ time.

4.3.3 Calculating Worst Case Complexity

From the previous section it can be noticed that the number of bends that are present in the bus are vital in the complexity analysis. Hence it becomes necessary to compute the worst case scenario. The architecture of the PR-Mesh allows the buses to bend at every opportunity and form a meandering structure as shown in Figure 15. At each processor the bus can be bent twice at the most. Hence allowing the maximum number of bends, the bus across an 8 x 8 mesh of processors looks as shown in Figure 15.

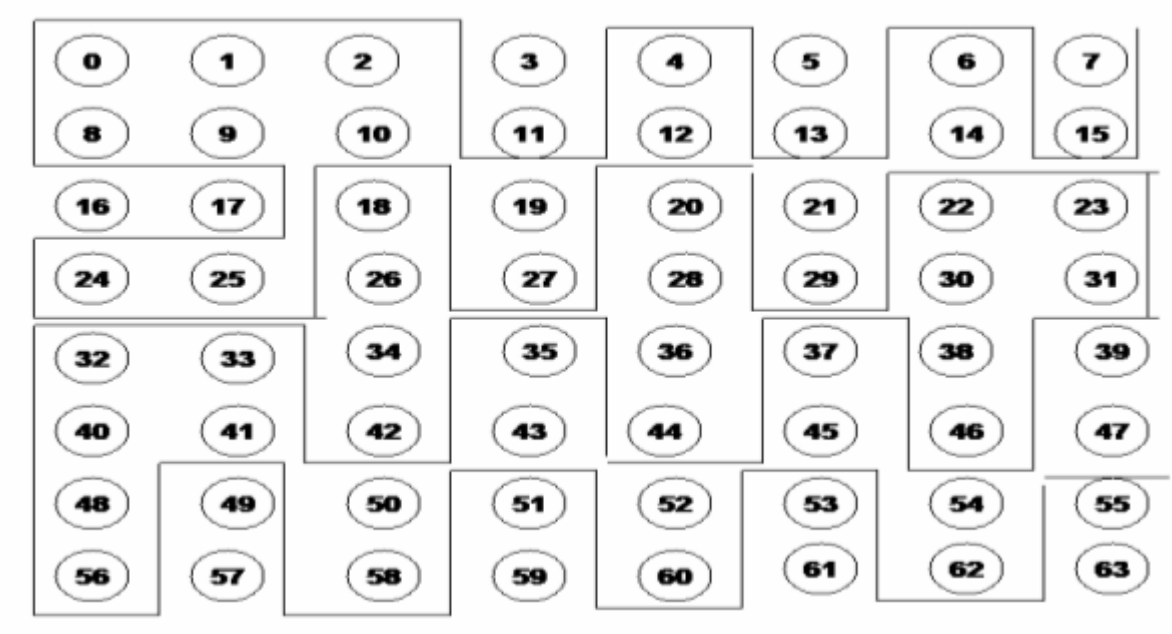


Figure 15 : Maximum Bends in an 8 x 8 Processor PR-Mesh

From the Figure 15 it can be noticed that there are about 58 bends in a 64 processor PR-Mesh which can be roughly thought of as $O(N^2)$ 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, $b <<< O(N^2)$ where b denotes the number of bends the bus can have.

Hence the worst case complexity of Simulating a PR-Mesh model on LARPBS (bus with multiple bends) is $O(\log \log N + \log N^2)$.

Lemma 3: Each step of an *M* x *M* processor PR-Mesh, in which each processor is connected to at most one bus and the bus can have more than one bend, can be simulated by an *N* (where *N* = *M* x *M*) processor LARPBS in $O(\log \log N + \log N^2)$ time.

4.4 *A note on simulating processors on multiple buses*

In the following section the underlying assumption is that the processors can be on multiple buses or can be the head of multiple buses as shown in the following Figure 16.

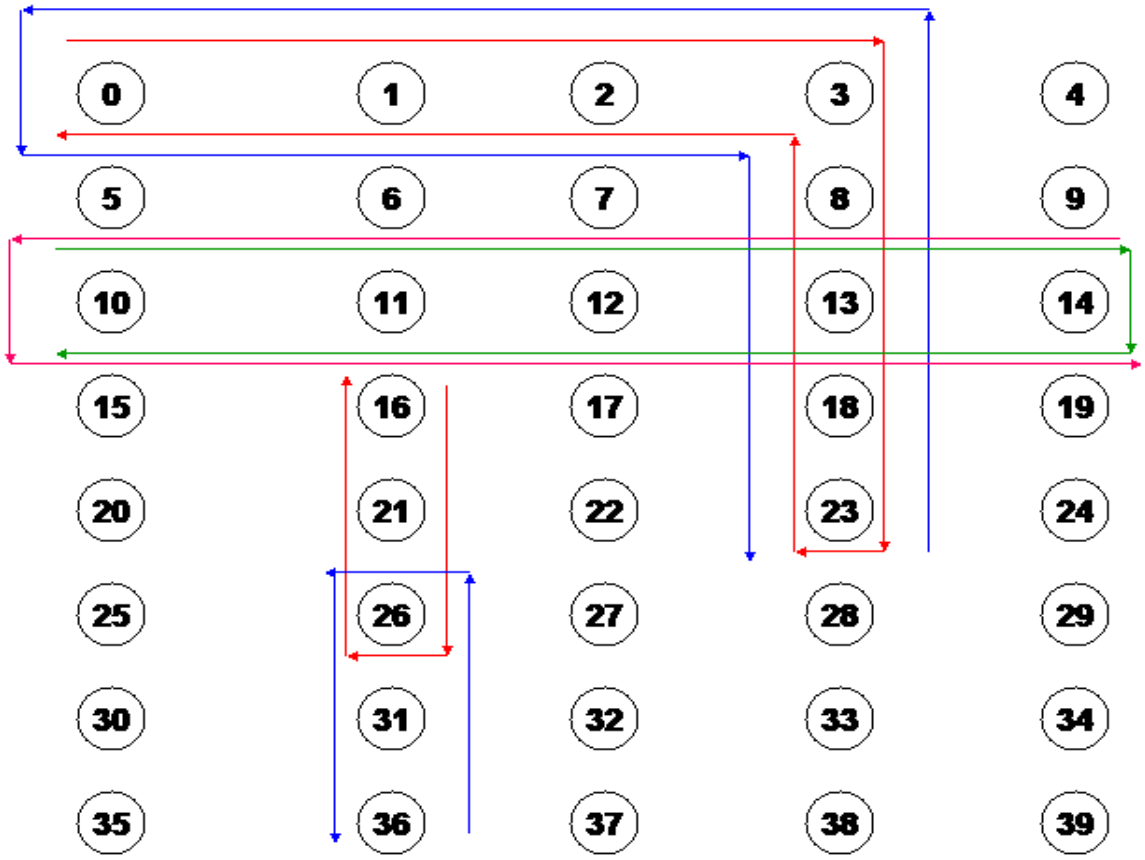


Figure 16 : Processors on PR-Mesh on Multiple Buses

From Figures 16 and 17, it is evident that unlike the simulation of processors on a single bus simulating processors that are on multiple buses is much more complicated by the fact that there are many more possible bus configurations that are possible as shown in Figure 17 and hence cannot be simulated by the same number of processors. It will further be explained why the simulation of such processors cannot be equivalent within a constant factor of processors.

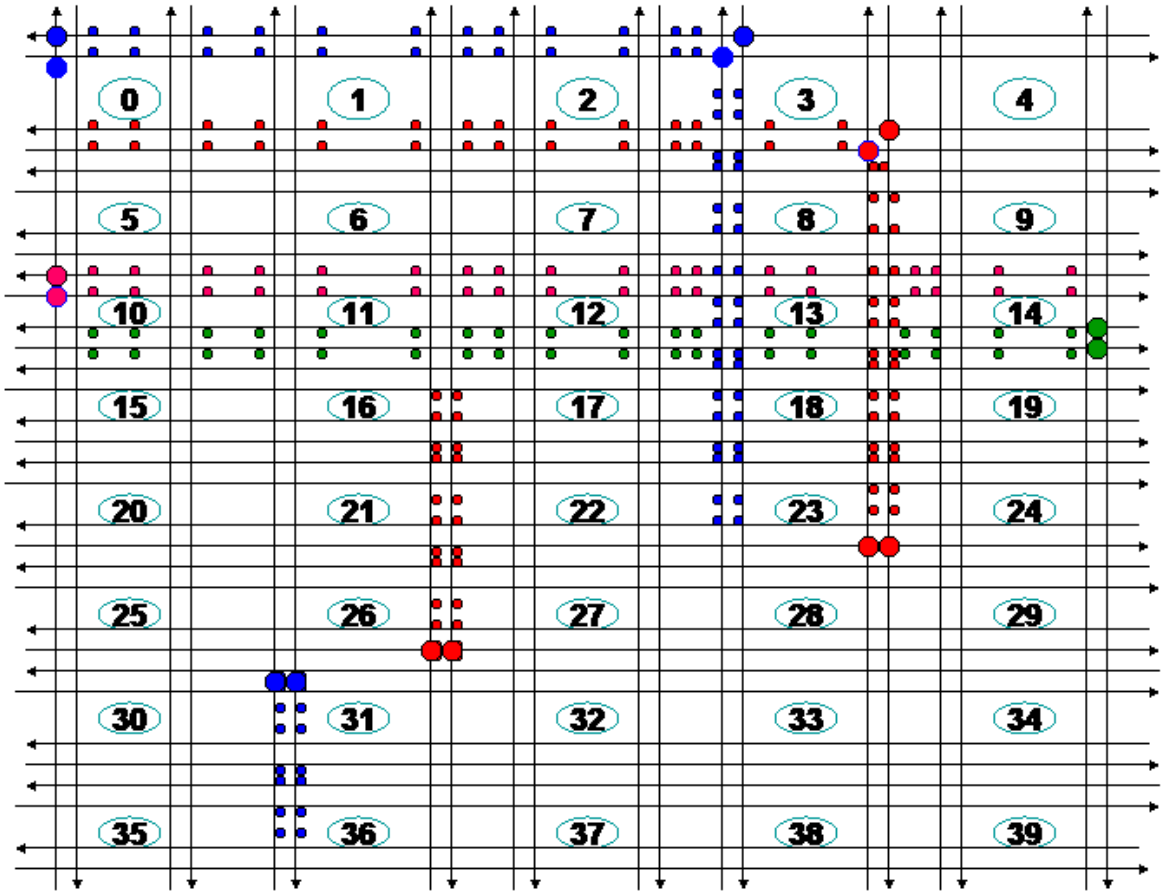


Figure 17 : Bus and Port Configurations of Processors on Multiple Buses

This kind of assumption for an increase in the number of processors where there is a large bus configuration due to multiple dimensions is already done in the simulation of cycle free linear reconfigurable mesh (CF-LR) Mesh - LR-Mesh, between the PR-Mesh - APPBS and relation between the PR-Mesh and the AROB [14], and hence is permissible. In the simulation of a two dimensional PR-Mesh on an LARPBS the increase in the number of processors is constant instead of the polynomial increase as in the simulations in [14].

Elaborating on the reason why there is a need for an increase in the number of processors from Figure 16 it can be seen that the processor P_{26} is the head of two buses and many other processors are on multiple buses. Hence within a single bus cycle, these processors might have to

function as members of different components which is not possible. This can be visualized as seen in Figure 18.

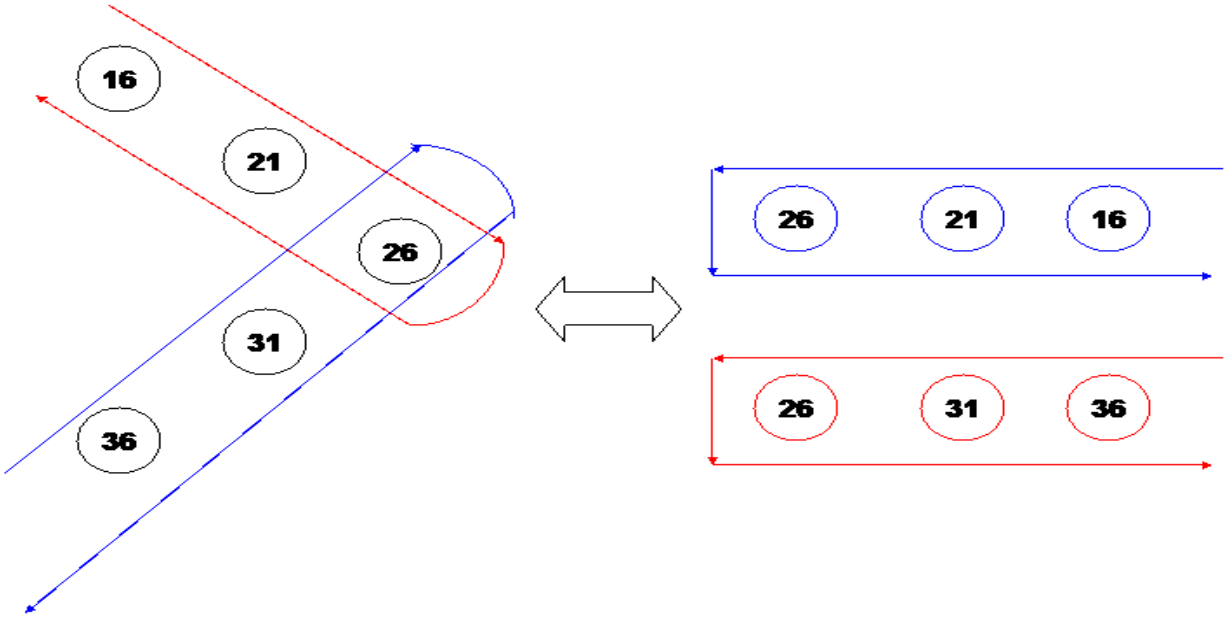


Figure 18 : Separating Processors on Multiple Buses

There has to be an increase in the number of processors so that they can be accommodated on as many as four buses [which is the maximum] like processor P_{13} at the same time. Hence the increase in the number of processors is constant, i.e. each processor has four copies rather than a polynomial increase. A notation has to be introduced prior to the presenting of 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. But 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 with which *N* has to be multiplied is

four. The equation is now modified as $F = Z(T, 4(N))$. Now all that is to be done is to determine the time needed to simulate processors that are on multiple buses on a PR-Mesh using an LARPBS.

For the simulation of processors on multiple buses on PR-Mesh the number of processors simulating them on the LARPBS has quadrupled. Hence the first step in the simulation is the indexing of processors and then arrangement or mapping on the LARPBS. The four copies of the processor P_i have indices P_{ia}, P_{ib}, P_{ic} and P_{id} respectively. For example P_5 on the PR-Mesh has four copies on the LARPBS with indices P_{5a}, P_{5b}, P_{5c} and P_{5d} respectively. Processor with index P_{ia}, is deemed as the "*master processor*" which holds the port and switch configurations of Processor P_i and the rest of the three processors are the slave processors at the beginning of the simulation. All processors with P_{ix} index are grouped together in ascending order of i as shown in the Figure 19 assuming there were three processors on the PR-Mesh.

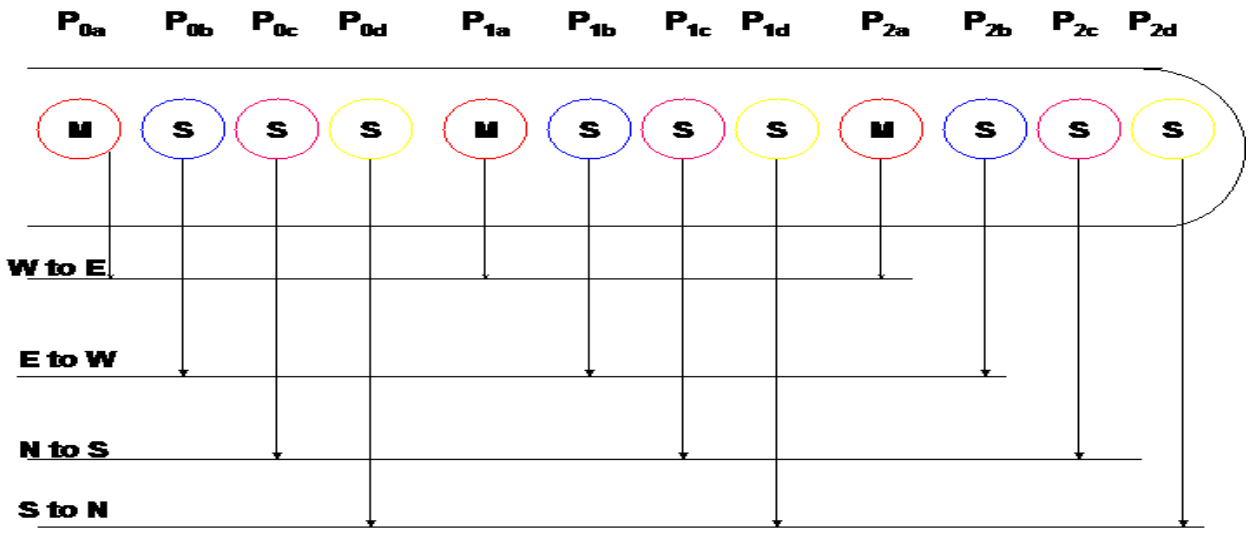


Figure 19 : Pre-Processing Phase of Simulation of Processors on Multiple Buses

During the pre-processing phase (which is the phase before the start of simulation) if the processor P_{ia} notices that it is on multiple buses it allows itself to simulate the bus segment in which the transmission is from left to right, it makes processor P_{ib} to simulate the bus segment in which the transmission is from right to left, it makes processor P_{ic} to simulate the bus segment in which the transmission is from north to south and it makes processor P_{id} to simulate the bus segment in which the transmission is south to north. That is respective port and switch configurations are passed on to these processors in constant time. After this step the processors can independently operate in each step of the simulation 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. For example processor P_{26c} and P_{26d} should be ranked consecutively.

4.5 *Simulating a PR-Mesh model on an LARPBS – Case 2(a)*

In this case the simulation is complicated by the fact that processors are on multiple buses. But during the preprocessing phase all those clusters have been separated out into individual segments. Here the main problem involves the elimination of duplicate processors that are present in certain segments. In this simulation the elimination process is discussed in detail while the rest of the simulation remains the same as in case 1(a).

Overview:

This section describes the high level operations needed to be performed for this simulation.

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
```

Pseudocode - Case 2(a)

Simulation:

The following section describes the actual simulation process with details about each step described in the overview.

Model: A $4N$ processor LARPBS [where $N = M \times M$] after the pre-processing phase has been completed.

Input: An $M \times M$ PR-Mesh

Output: Processors in a bus grouped together in the order in which they lie on PR-Mesh

Assumptions: For simulating the PR-Mesh on LARPBS the following assumptions are made

1. Each Processor on PR-Mesh is connected to any number of the four buses.

2. Each bus has just one bend.

Steps:

Begin

1. All processors which are the head of a segment set flag as 1.
2. Compress all processors holding flag value as 1.
3. Compute the prefix sum of each of these processors; this denotes the ranks of each processor (B_{RANK}).
4. Forming Bus Sections
    a. Arrange all P_{ia} processors together in the increasing order of i such that they form Group-a. Similarly group P_{ib} processors, P_{ic} processors and P_{id} processors to form Group-b, Group-c and Group-d respectively in row major order. In other words, Group-a consists of processors with index P_{ia} only where i ranges between 0 and N. And all the N^{th} processors with in the group set their segment switches to form 4 different sub-arrays which will be named Group-a, Group-b, Group-c and Group-d.
    b. Processors in Group-c and Group-d are then arranged in column major format. Now two among the four sub-arrays is in row major order and other two in column major order.
    c. Processors in Group-a simulate bus segments in which the transmission is from west to east. Processors in Group-b simulate bus segment in which the transmission is from east to west. Processors in Group-c simulate bus segments in which the transmission is from north to south and Processors in Group-d simulate bus segments in which the transmission is south to north.
5. Forming row and column segments and ranking row/column only bus

a. All the processors in Group-a and Group-b, whose East-West ports are not connected as shown in Figure 5(f) set their segment switches to cross. All processors in Group-c and Group-d whose North South ports are not connected as shown in Figure 5(b) set their segment switches to cross.

b. All the processors in Group-a and Group-b, that segmented in the previous step send their index to its left/right neighbor processor as shown in the Figure 11(a). The messages are denoted by the arrows below the processors. [Left neighbor is processor sending its index – 1, and right neighbor is processor sending its index +1]. All disconnected processors also do the same in these two groups. All processors in Group-c and Group-d who segmented in the previous step send their index to its left/right neighbor processor. [Left neighbor is processor sending its *index – M* and right neighbor is processor sending its *index + M*]. All disconnected processors also do the same in these two groups.

c. If a processor receives a message, it must set 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 or a column bus therefore it cannot also lie on the row or a column bus that the receiving processor lies on.

d. The head and tail segment processors with in each segment of each group exchange their indices and also the head of the segment informs all the processors between itself and the tail of the head and the tail processor indices.

e. One of the processors on either end of the segment is head of the bus. It sends a value of 1 as (H_{SEG}) to the other processor that lies on the other end of the segment and also the B_{RANK}.

f. All processors find their temporary ranks based on the direction of transmission. For segments where the head of the bus has been identified, this is the final rank of the processors. For example the processors P_{10} to P_{14}, P_{16} to P_{26}, and P_{26} to P_{31} from Figure 16 find their ranks and also the bus rank at the end of this step. For other processors the following steps are continued.

6. Ranking Processors on buses with a bend

   a. All Processors straighten their segment switches to form a single LARPBS.

   b. 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 20. It can be said with certainty that one among the two mirror pivots will definitely 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. It must be noted that all processors know the id as well as index (when arranged in terms of groups a, b, c and d) of the processor simulating its pivot. Thus multiple one-to-one communications can take place in a single step.

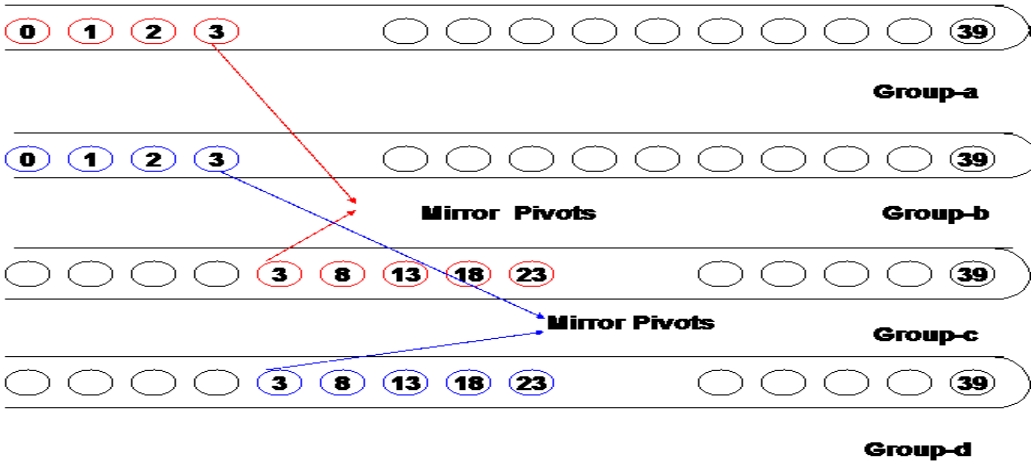


Figure 20 : Simulation of Processors on Multiple Buses

c. Only one among the two mirror pivot possessors is ranked and it is the communicating processor that knows the bus rank is ranked while the other one becomes idle after passing on the information.

d. The adjustment of the ranks is similar to case 1(a).

7. All disconnected processors, idle processors and head of segments holds flag value of 1 and are compressed.
8. Head of segments hold total number of processors in their segments (P_{TOTAL}), while the disconnected processors and idle processors that do not belong on a bus hold a value of one (as they are a single entity within their segments) and a prefix sum ($N_{SLOT\text{-}START}$) is computed for each segment.
9. Form row and column segments again and broadcast the $N_{SLOT\text{-}START}$ value.
10. Each processor computes its new index as $N_{SLOT\text{-}START} + P_{RANK.}$
11. Each processor then arranges itself according to the new index.

End

4.5.1 Complexity Analysis

The following section describes in detail the complexity analysis of the algorithm discussed in this section. The steps discussed below take into account the complexity of every step of the algorithm and give a final value based on the summation of these steps.

The complexity of steps in the simulation algorithm is as follows:

1. Compression algorithm takes $O(1)$ time [2].
2. For processors to compute their temporary ranks along a row or column bus takes $O(1)$ time [rank = *index of head of segment – index of the processor computing its rank*] [2].
3. 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].
4. All communication between processors [this includes communication between two processors, multicasting or broadcasting] takes $O(1)$ time [2].
5. All the internal functions that the processors perform for e.g. adjusting their ranks once the other processors along the bus, finding number of processors along their segments, etc. takes $O(1)$ time [2].
6. The prefix sum is computed for the head of the segments. For integers with bounded magnitude algorithm for prefix sum computation, takes $O(\log \log N)$ time using N processors [12].
7. The permutation routing of the processors in LARPBS takes $O(1)$ time [11].

   This proves that PR-Mesh (T, M x M) $\subseteq$ LARPBS ($O(\log \log N)$, $4N$)

From the notation above it is to be understood that any class of problems solved by the PR-Mesh in $O(T)$ time steps using M x M processors can be solved by an LARPBS in $O(\log \log N)$ time using $4N$ processors.

Lemma 4: Each step of an *M* x *M* processor PR-Mesh, in which each processor can be connected to multiple buses where the bus can have at most one bend can be simulated by an LARPBS in *O*(log log *N*) time using 4*N* (where *N* =*M* x *M*) processors.

4.6 *Simulating processors on multiple buses with multiple bends*

In this section the more general scenario that many processors can be on multiple buses and the buses are likely to have multiple bends is simulated. An example is as shown in Figure 21.

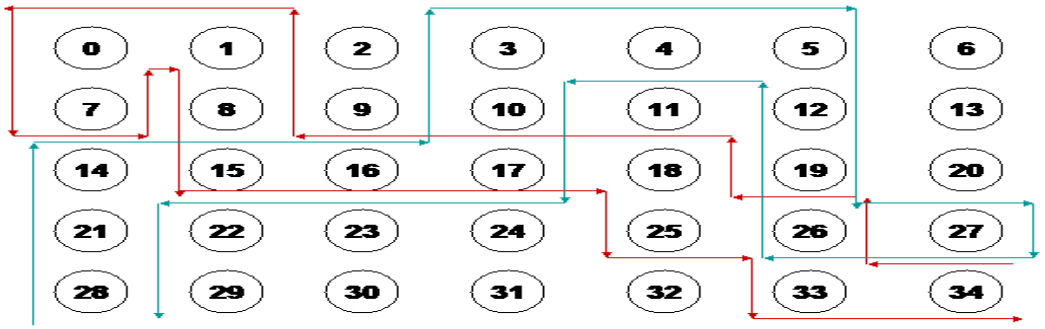


Figure 21 : Processors on Multiple Buses with Multiple Bends

Figure 21 shows in detail how the processors can exist on multiple buses. For example processor P_{26} is a pivot processor for two separate buses which are shown in different colors. Since the preprocessing phase of the simulation already separates out the processors into different segments based on the directionality of the simulation the pivots can communicate with different processors with in the same bus cycle. From the figure above and the simulation of processors on multiple buses as discussed in case 2(a) 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 then ranking processors on the bus. Since after the identification and elimination of the mirror pivots makes the problem same as processors on single bus with multiple bends the rest of the simulation is done as discussed in Case 1(c). Hence only the first part of the algorithm is discussed here.

4.7 *Simulating a PR-Mesh model on an LARPBS – Case 2(b)*

This scenario of the simulation is the most probable and realistic scenario to be considered. Similar to case 2(a) it is assumed that the processors are on multiple buses which are bent multiple times. Again all those clusters have been separated out into individual segments during the preprocessing phase. Here again the main problem involves the elimination of duplicate processors that are present in multiple segments. In this simulation the elimination process is discussed in detail while the rest of the simulation remains the same as in case 1(c).

Overview:

This section describes the high level operations needed to be performed for this simulation.

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

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

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
    In segments where bus rank not know
        Preserve and rank pivot farther in direction of transmissi
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 (denotes total processors on bus)
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 - Case 2(b)

Model: A 4*N* processor LARPBS [where $N = M \times M$] after the pre-processing phase has been completed.

Input: An $M \times M$ PR-Mesh

Output: Processors in a bus grouped together in the order in which they lie on PR-Mesh

Assumptions: For simulating the PR-Mesh on LARPBS the following assumptions are made

1. Each Processor on PR-Mesh is connected to any or all or none of the four buses.
2. Each bus can have multiple bends.

Steps:

Begin

1. All processors which are the head of a segment set flag as 1.
2. Compress all processors holding flag value as 1.
3. Compute the prefix sum of each of these processors; this denotes the ranks of each processor (B_{RANK}).
4. Forming Bus Sections
    a. Arrange all P_{ia} processors together in the increasing order of i such that they form Group-a. Similarly group P_{ib} processors, P_{ic} processors and P_{id} processors to form Group-b, Group-c and Group-d respectively in row major order. In other words, Group-a consists of processors with index P_{ia} only where i ranges between 0 and N. And all the N^{th} processors with in the group set their segment switches to form 4 different sub-arrays which will be named Group-a, Group-b, Group-c and Group-d.
    b. Processors in Group-c and Group-d are then arranged in column major format. Now two among the four sub-arrays is in row major order and other two in column major order.

c. Processors in Group-a simulate bus segments in which the transmission is from west to east. Processors in Group-b simulate bus segment in which the transmission is from east to west. Processors in Group-c simulate bus segments in which the transmission is from north to south and Processors in Group-d simulate bus segments in which the transmission is south to north.

5. Forming row and column segments and ranking row/column only bus

   a. All the processors in Group-a and Group-b, whose East-West ports are not connected as shown in Figure 5(f) set their segment switches. All processors in Group-c and Group-d whose North South ports are not connected as shown in Figure 5(b) set their segment switches.

   b. All processors in Group-a and Group-b who segmented in the previous step send their index to its left/right neighbor processor .The messages are denoted by the arrows below the processors. [Left neighbor is processor sending its *index* – 1, and right neighbor is processor sending its *index* +1]. All disconnected processors also do the same in these two groups. All processors in Group-c and Group-d who segmented in the previous step send their index to its left/right neighbor processor. [Left neighbor is processor sending its *index* – *M* and right neighbor is processor sending its *index* + *M*]. All disconnected processors also do the same in these two groups.

   c. If a processor receives a message, it must set 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 or a column bus therefore it cannot also lie on the row or a column bus that the receiving processor lies on.

d. The head and tail segment processors with in each segment of each group, exchange their indices and also the head of the segment informs all the processors between itself and the tail of the head and the tail processor indices.

e. One of the processors on either end of the segment is head of the bus. It sends a value of 1 as (H_{SEG}) to the other processor that lies on the other end of the segment and also the B_{RANK}.

f. All processors find their temporary ranks based on the direction of transmission. Segments where the head of the bus has been identified this is the final rank of the processors. For other processors the following steps are continued.

6. Identification and Elimination of Mirror Pivots

a. All processors straighten their segment switches to form a single LARPBS.

b. In the case that the bus has multiple bends, the bus with mirror image of the same pivot processor is present in two of the bus segments. Hence one among the two needs to be eliminated. Elimination here means not ranking one of the processors. 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. Since the direction of the transmission is known the mirror-pivot farther in the direction of transmission is always chosen and the other pivot informs the chosen one the number of processors in its segment and also the identity of the pivot that it needs to communicate in the next steps. For example on the red bus processor P_1 will have mirror pivots in P_{1b} and P_{1d} and P_{1d} is chosen as its farther in the direction of communication

c. After this step all the segments have been joined together and now the simulation for the ranking is similar to Case 1(c). Once ranking of the components 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.

End

4.7.1 Complexity Analysis

The following section describes in detail the complexity analysis of the algorithm discussed in this section. For this simulation the first few steps are common to the previous sections. The elimination of the mirror pivot processors is just a communication step and takes $O(1)$ time [2]. The prefix sum is computed for the head of the segments. For integers with bounded magnitude algorithm for prefix sum computation, takes $O(\log \log N)$ time using N processors [12] while the ranking of the processors takes $O(\log b)$ steps [1] where b denotes the number of bends in the bus. The worst case is similar to Case 1(c). Hence the total simulation time takes $O(\log \log N + \log b)$ steps.

This proves that PR-Mesh $(T, M \times M) \subseteq$ LARPBS $O(\log \log N + \log b), 4N)$

From the notation above it is to be understood that any class of problems solved by the PR-Mesh in $O(T)$ time steps using $M \times M$ processors can be solved by an LARPBS in $O(\log \log N + \log b)$ time using $4N$ processors.

Lemma 5: Each step of an *M* x *M* processor PR-Mesh, in which each processor is connected to multiple buses where the buses can have multiple bends (b) , can be simulated by an 4*N* (where *N* = *M* x *M*) processor LARPBS in *O*(log log *N* +log *b*) time or *O*(log log *N* +log N^2) in worst case.

5 CONCLUSION AND FUTURE RESEARCH DIRECTION

From the simulation it is established that a two dimensional M x M PR-Mesh can be simulated on an N or $4N$ (depending on the bus configuration) processor LARPBS (where $N = M$ x M). It has to be noted that the PR-Mesh is slightly more powerful model than the LARPBS due to the much richer configurations that are possible due to a higher dimension.

The results are presented below in Table 2 for the different cases that were considered for our simulations. The results are tabulated based on different cases that were considered and on the assumptions that were made for each of those cases. The table also lists the number of processors that were needed for that particular case and the time taken for the simulation. The "*WC*" in the table indicates the worst case scenario where the bus bends (b) multiple times.

Table 2 : Results of Simulation

	Assumptions	**No. of Processors**	**Time Taken**
I.	Processors on a Single Bus with Single Bend	N	$O(\log \log N)$
II.	Processors on a Single Bus with Multiple Bends	N	$O(\log \log N + \log b)$ **WC:** $O(\log \log N + \log N^2)$
III.	Processors on Multiple Bus with Single Bend	$4N$	$O(\log \log N)$
IV.	Processors on a Multiple Bus with Multiple Bends	$4N$	$O(\log \log N + \log b)$ **WC:** $O(\log \log N + \log N^2)$

Instead of just considering one general scenario for the simulation we have considered different cases. These scenarios or cases 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. Another point worth mentioning at this point is that the efficiency of the simulation directly depends on the efficiency of the prefix sum computation for integers with bounded magnitude.

This simulation is first of its kind to establish a relationship between a one dimensional optical model 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 mapping of processors from the PR-Mesh to the LARPBS was done successfully by preserving the order in which the processors appeared on the two dimensional PR-Mesh. The aim of the simulation was achieved by making the processors communicate on a linear bus instead of a two dimensional bus. The complexity in reconfigurable architecture is due to two main factors. One is due to the functionalities provided by the models and another due to the complexity of the bus structure. In this case the functionalities provided by both the 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 separate sub-arrays in the LARPBS, a processor that is a part of multiple buses may have to communicate with processors in different sub – arrays with in a single bus cycle.

Since the PR-Mesh is a two dimensional extension of the LARPBS there was a natural correspondence between them that was exploited, but there are many other models which have much richer switch and port configurations or due to the 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 (for example the AROB and the LAROB] and so on. Similar to the PR-Mesh, the LR-Mesh allows no branching and forms linear buses and hence it is possible to simulate the LR-Mesh on the LARPBS.

On a different note, it has to be remembered that the PR-Mesh is in fact a *k*-dimensional model as depicted in the Figure 22. Figure 22 represents the structure of a two dimensional PR-Mesh that was utilized in this simulation and the three dimensional structure that is to be considered for the future simulation purposes. This can then lead to future work in expanding the simulation to the *k* – dimensional PR-Mesh model. The simplicity of the two dimensional model is that there are only two axes to be considered. But with an increase in the number of dimensions the complexity of the bus structure will increase.

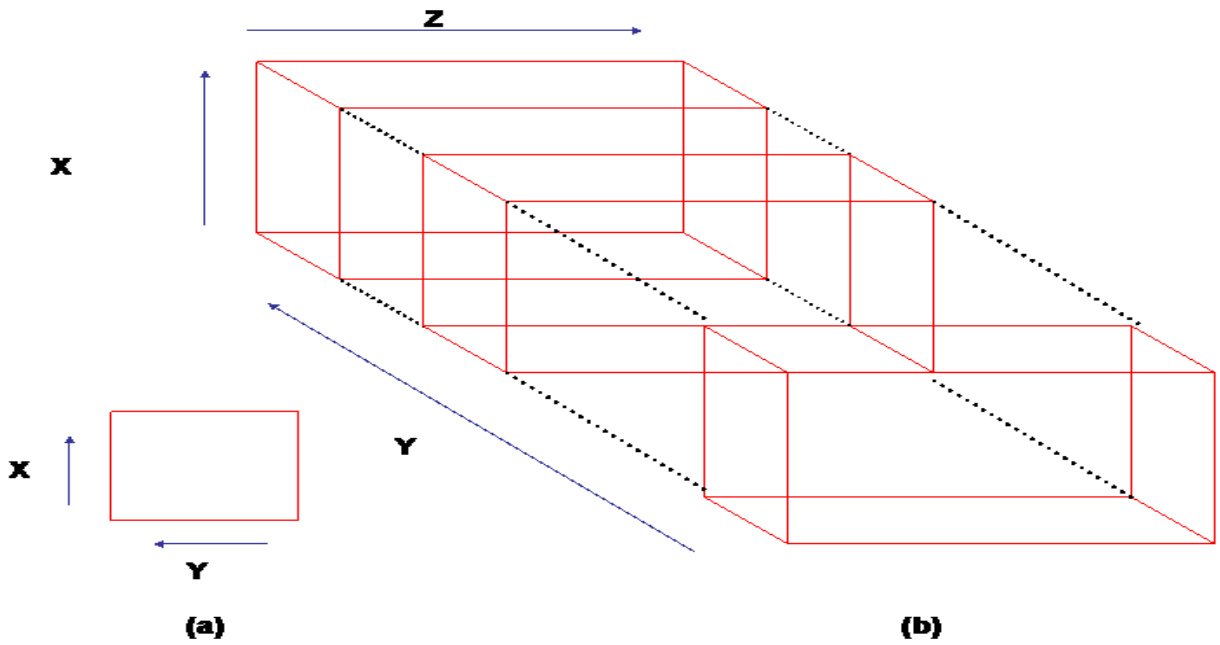


Figure 22 : PR-Mesh (a) Two Dimensional PR-Mesh (b) Three Dimensional PR-Mesh

The simulation that we completed so far on the LARPBS is only for the two dimensional version of this model. Hence a much more generalized version of algorithm that is capable of simulating for any value of *k* is to be developed. Some of the areas where some thought needs to be put in 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 found. But the process of identification of different bus segments is complicated by the presence of multiple dimensions. Similarly a simulation involving models in which cycles are permissible should be looked into as well.

From this thesis, we can now easily relate the complexity of the LARPBS to that of the PR-Mesh. This 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 processor the same as the simulated model and investigate the time taken.

BIBLIOGRAPHY

[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.* 106 (1998), 237-258.

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

[4] Y. Pan, “Order statistics on optically interconnected multiprocessor systems”, *Opt. Laser Tech*. 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”, to appear in *Informatica*.

[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] Amitava Datta, “Multiple Addition and Prefix Sum on a Linear Array with a Reconfigurable Pipelined Bus System”, *The Journal of Supercomputing*, 29, 303–317, 2004.

[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] 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.

[15] 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.

[16] Mounir Hamdi,Chunming Qiao,Yi Pan, and J. Tong, “Communication-Efficient Sorting Algorithms on Reconfigurable Array of Processors With Slotted Optical Buses”, *Journal of Parallel and Distributed Computing* 57, 166-187 (1999).

[17] L. Chen, Y. Pan, and X. Xu, "Scalable and Efficient Parallel Algorithms for Euclidean Distance Transform on the LARPBS Model," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 15, No. 11, Nov. 2004, pp. 975-982.

[18] L. Chen, Y. Pan, Y. Chen, and X. Xu, ``An Efficient Parallel Algorithm for Euclidean Distance Transform," *The Computer Journal* , Vol. 47, No. 6, 2004, pp. 694-700.

[19] L. Chen and H. Chen, Y. Pan, and Y. Chen, "A Fast Efficient Parallel Hough Transform Algorithm on LARPBS," *The Journal of Supercomputing*, Vol. 29, pp. 185-195, 2004.

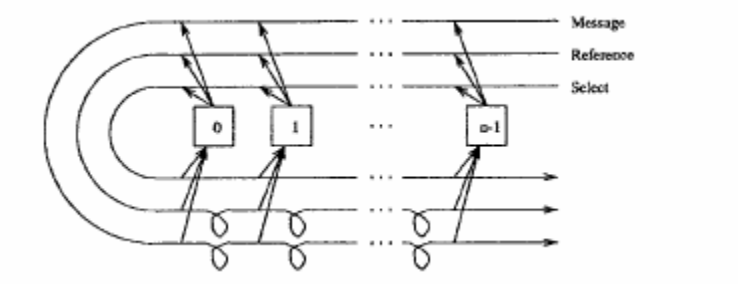
[20] S.-J. Horng, H.-R. Tsai, Y. Pan, and J. Seitzer, ``Optimal Algorithms for the Channel-Assignment Problem on a Reconfigurable Array of Processors with Wider Bus Networks," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 13, No. 11, November 2002, pp. 1124-1138.

[21] Y. Pan, Y. Li, J. Li, K. Li, and S.Q. Zheng, "Efficient Parallel Algorithms for Distance Maps of 2D Binary Images Using an Optical Bus," *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans* , Vol. 32, No. 2, March 2002, pp. 228-236.

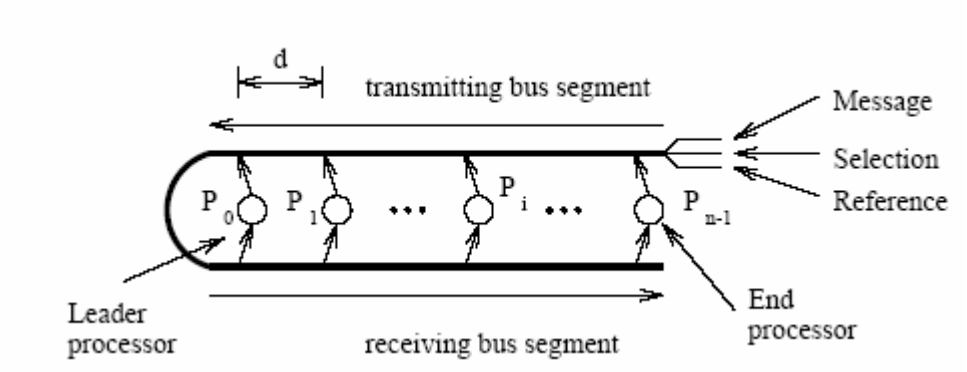
APPENDIX

Optical Reconfigurable Models

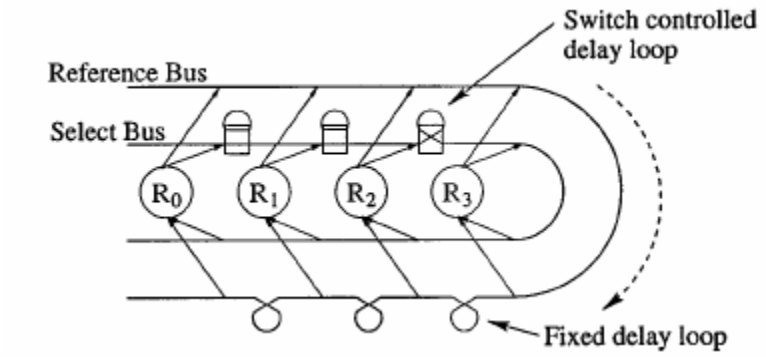
1. Model of POB [3]



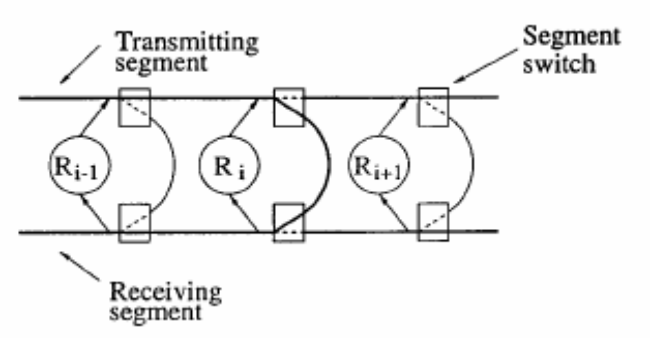
2. Model of one dimensional APPB [6]



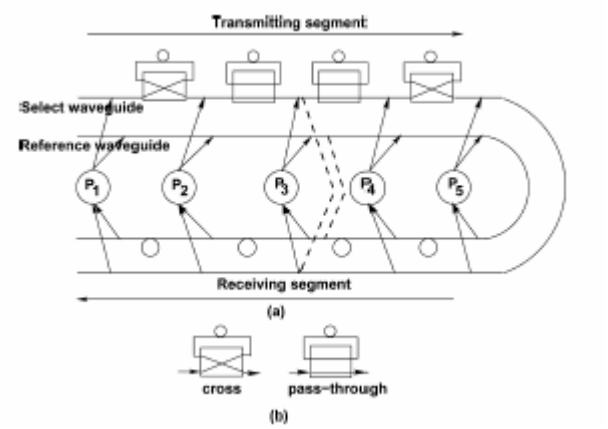
3. Model of LPB and LARPBS [11]



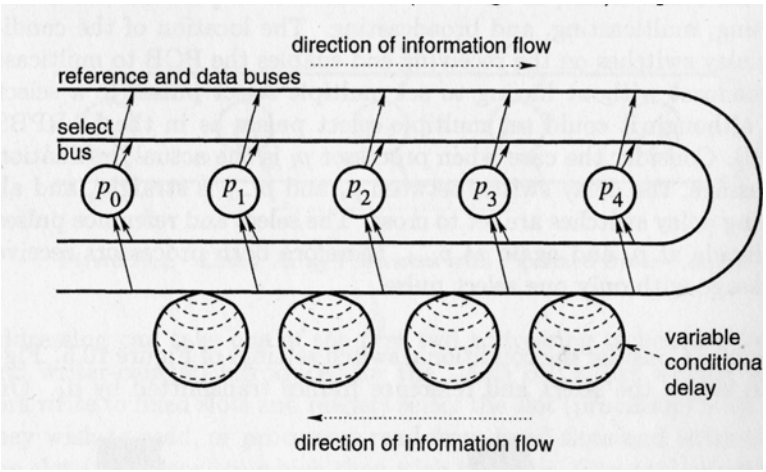
4. Segment Switches on an LARPBS [11]



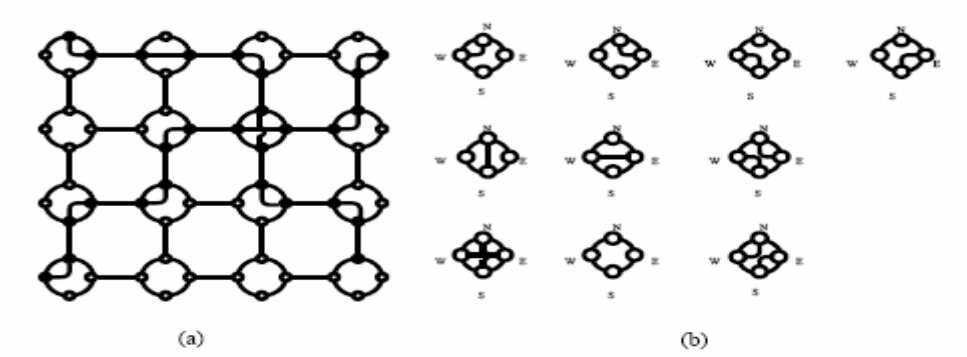
5. Model of LARPBS with Switch Connections [12]



6. Model of LAROB [1]

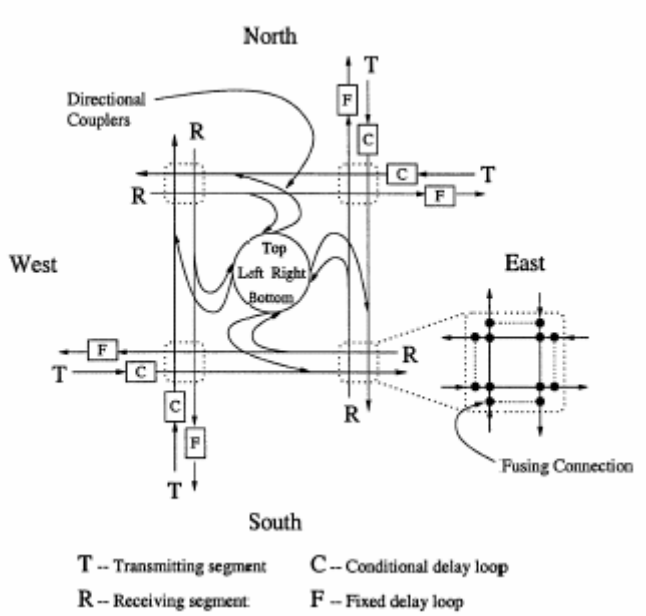


7. Model of AROB [6]

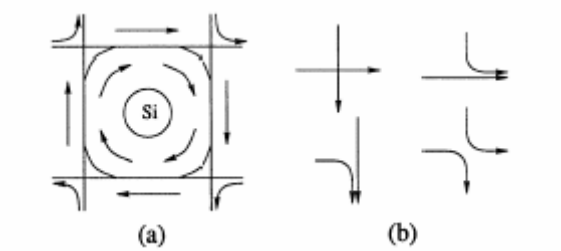


(a) Two-Dimensional Reconfigurable Network (b) Switch Configurations

8. Model of PR-Mesh [5]

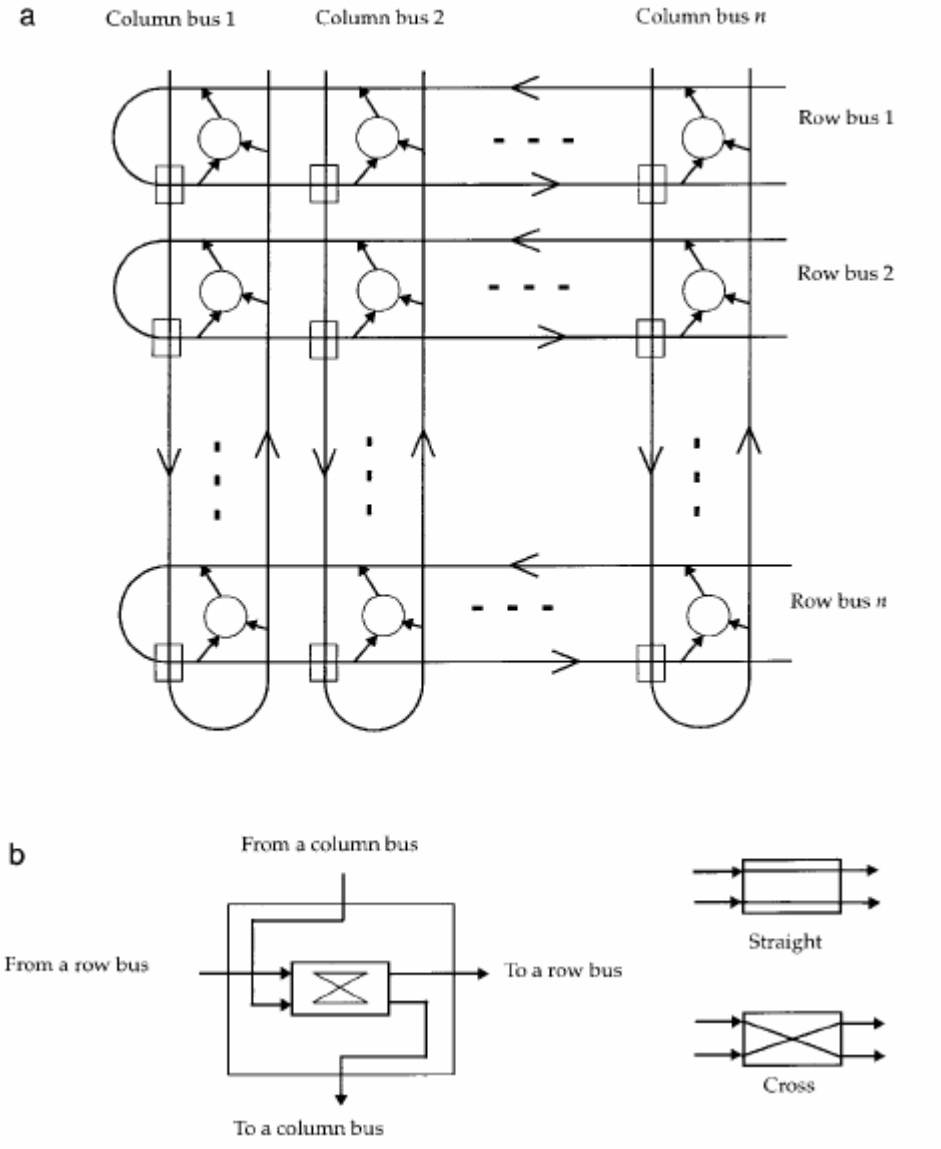


9. Model of APPBS with Switches [1] [14]



(a) switch connections at each APPBS processor (b) switch configurations at each processor

10. Model of RASOB



(a) RASOB architecture (b) Switch connecting row and column bus