Models using Buses

Chapter 10

Introduction

- Mesh Advantages
  - Constant link length.
  - Easy to expand.
  - Large bisection width (nr. of wires that must be cut to divide the network into two equal parts).
  - Small and a fixed number of connections per PE.
  - Models 2-D world well (and 3-D reasonably well).
- Disadvantages: diameter is large.
- Chapter 8 and 9 solutions

- Replace mesh connections with faster connections
  - e.g., either of the ”mesh of trees” (see Figure 2.11 & 2.12)
- Add new connections to existing connections.
  - e.g., pyramid
- Replace mesh with an architecture with a smaller diameter (e.g., hypercube, star).

- Disadvantages
  - New architectures are not as easy to expand.
    - e.g., number of connections to *each node* increases on hypercube
  - Physical length of links grow with the number of PEs in many architectures
    - Time to traverse longer links increases.
- Alternate Solution: Use bus-enhancements to reduce the diameter
  - Some or all PEs are attached to buses.

- Processors on the same bus can communicate directly.

- ***Fixed Bus Models***
  - Single Global Bus Model
    - The 2-D mesh architecture is included.
    - All PEs are connected to a single static bus.
    - A datum placed on the bus by one PE can be read by all other PEs.
      - i.e., is a broadcast
    - At any given time, only one PE can broadcast to the other PEs.
    - If more than one PE broadcasts, then an arbitrary one is selected by bus to succeed.
      - No standard assumption concerning results of a multiple broadcast.
      - Usually, programmer responsible for avoiding multiple broadcasts.
    - Example: See following Figure

10.1 from Akl's textbook:

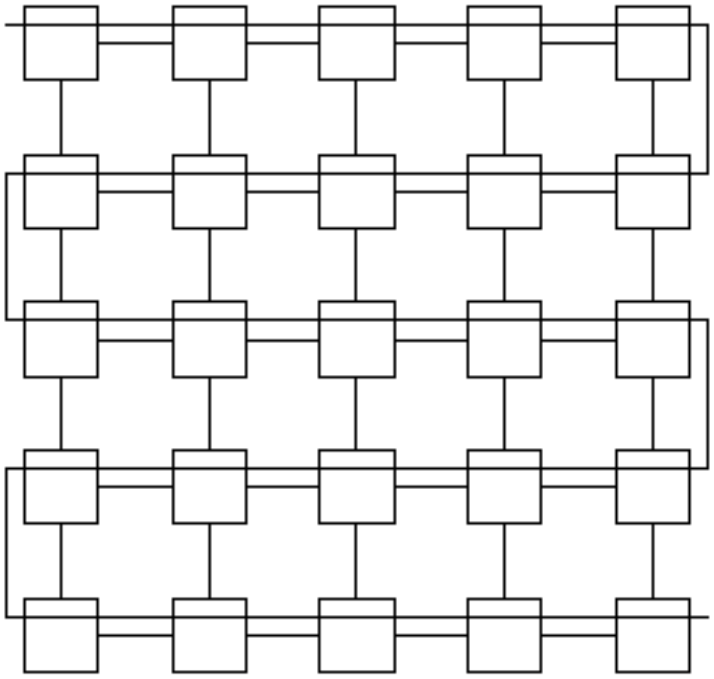


Figure 10.1: Mesh augmented with a global bus.

- Mesh with Multiple Buses (**MMB**)
  - All PEs in each row and column are connected by a bus.
  - A PE can broadcast datum to other PEs on either its row or column bus.
  - At each step, broadcasts can occur along one or more rows (columns).

- The row and column buses can not be used in the same step.
- Example: See Figure 10.2 from Akl's Textbook below:

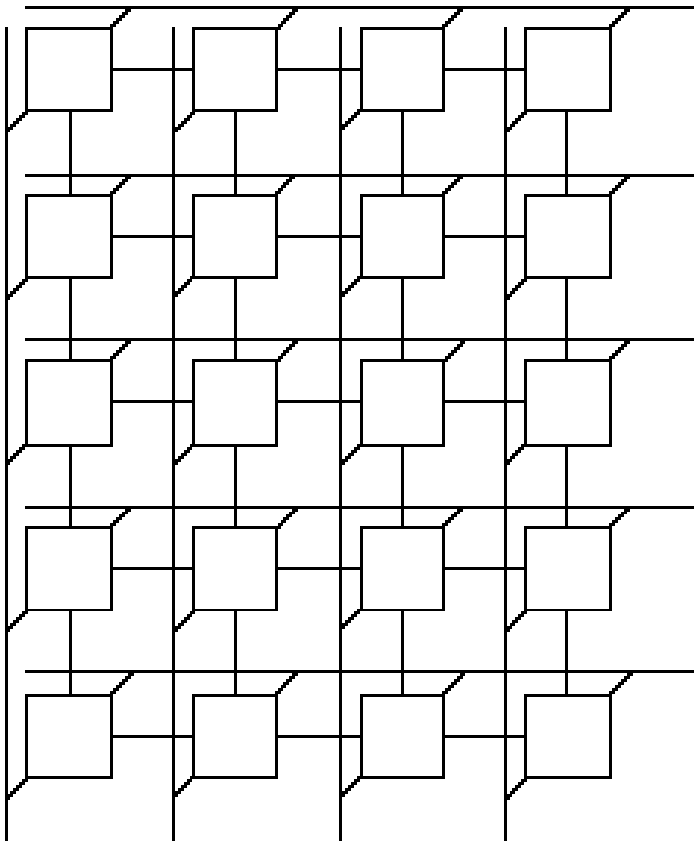


Figure 10.2: Mesh augmented with row and column

- Reconfigurable Bus Models
  - Allows buses to be created dynamically during the execution of an algorithm.

- The number, shape, and length of the buses is determined and changed by the algorithm.
- One PE can broadcast to all other PEs on its bus.

- Optical Bus Models of Computation
  - Differs from usual buses, which
    - are electronic
    - allow only exclusive broadcasts.
  - An optical bus allows multiple PEs to place their datum on it simultaneously.
- Traversal time for buses
  - Let $B(L)$ denote a bus of length L.
  - Let $T_{B(L)}$ denote the time for word-size datum to travel the length of a bus of length $B(L)$.
  - Travel time for electronic buses depends upon
    - Technology used to implement the bus
    - Length of bus
    - Bus capacity

- Material bus is made of, which determines the ”friction” on bus.
- Is typically linear on optical buses, due to speed of light.
- Some engineers argue that $T_{B(L)}$ should be assumed to be linear for all buses.
  - That is, $T_{B(L)} = cL$ for some constant c.
  - If implemented as a tree, then $T_{B(L)} = c(\log L)$
- Another possibility is to include $T_{B(L)}$ as a variable when expressing running times.

- **CLAIM:** It is reasonable to assume that $T_{B(L)} = O(1)$.
  - It is reasonable to assume that the number of PEs are not unbounded.
    - The human brain is estimated to have about 8 billion neurons.
    - New parallel models of

computation may be needed for computational systems approaching this size.

- If the number of PEs are assumed to be at most a few million, then $T_{B(L)}$ takes less time than O(1)-time operations such as addition and multiplication.
- As technology improves,
  - $T_{B(L)}$ should continue to decrease.
  - the length L of a bus needed to join a fixed nr of PEs should decrease.
- Argument that $T_{B(L)} = O(1)$ is similiar to argument in section 2.4 of Akl's textbook that the time to access a location in a memory of size M is $O(1)$.
  - Technically, can argue that $T_{B(L)}$ is $O(M)$ - or if implemented as a tree that $T_{B(L)}$ is $O(\log M)$

        - Practically, can show to be $O(1)$.

Finding a Maxima on a Mesh with Global Bus

- In Section 10.1.1, algorithm given for $\sqrt{n} \times \sqrt{n}$ mesh with global bus with $O(\sqrt[3]{n})$ execution time, which is best possible by Section 10.1.2
- **NOTE:** May add further details on this.

Finding a Maxima on MMB

- Algorithm uses the **Mesh Maximum Algorithm** for 2D mesh (pg 430-431 & Fig 10.3a in Akl).
    - Phase 1 of algorithm requires $\sqrt{n} - 1$ basic steps.
        - Initially, each neighbor in the rightmost column sends its data to its left neighbor
        - For $\sqrt{n} - 2$ additional steps, each processor P(i,j) receiving a datum from its right neighbor compares this datum to its own and

forwards the larger to its left neighbor.

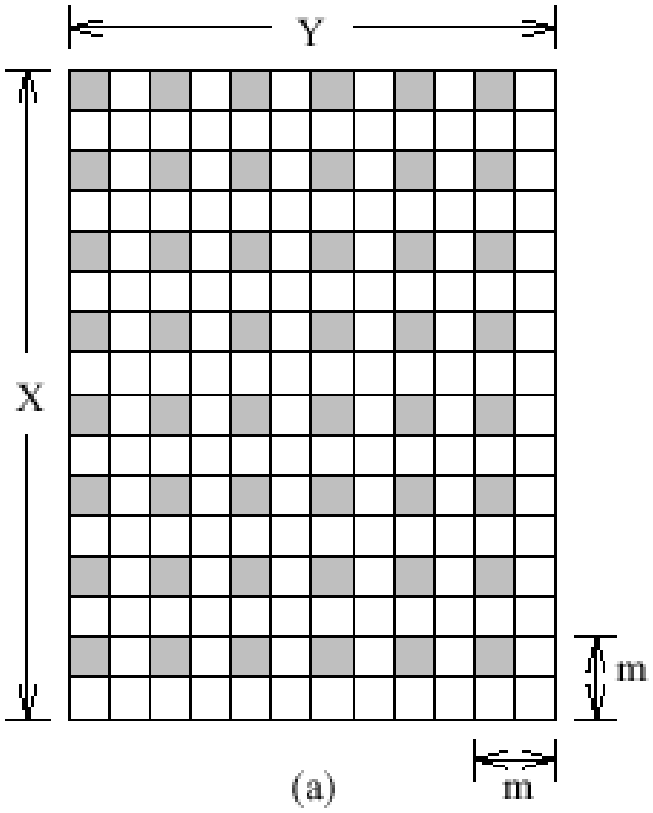
- After preceding steps, each processor $P(i,0)$ contains the maximum datum x_i in the i^{th} row.

- Phase 2 of algorithm also requires $\sqrt{n} - 1$ basic steps
  - Initially, $P(\sqrt{n} - 1, 0)$ sends the maximum of its row to its neighbor $P(\sqrt{n} - 2, 0)$ above it.
  - For $\sqrt{n} - 2$ additional steps, each processor P(i,j) receiving a datum from its lower neighbor compares this datum to its own and forwards the larger to its upper neighbor.

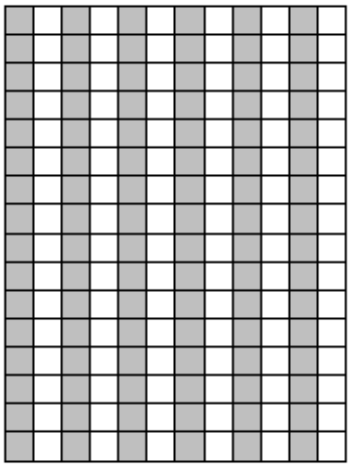
- Recall, in the MMB Architecture,
  - The standard mesh is augmented with row & column buses.
  - Processors can communicate using local links to four neighbors.
  - All processors connected to the same bus can read a value being broadcast

simultaneously.
  - A value can be broadcast to all PEs in 2 steps.
- Preliminaries for Algorithm
  - Let n data items be stored in an $X \times Y$ MMB with $n = XY$.
  - For sake of definiteness, assume $X \geq Y$.
  - The algorithm partitions the mesh into $m \times m$ blocks.
  - For ease of presentation, **assume** X is a multiple of m and Y is a multiple of m^2.
  - The value of m is optimized after the algorithm is given.
  - A row of $m \times m$ blocks is called a **band.**
- ***Algorithm: MMB Maximum***
  - Following are summary of algorithm steps from Akl textbook (pg 435-438)
- **1.** Use Mesh Maximum Algorithm for 2D mesh discussed earlier to find

the maximum in each $m \times m$ block.

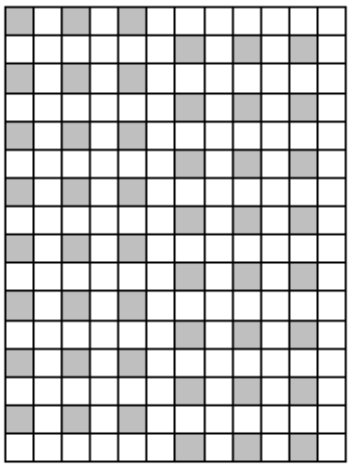


2. Copy maximum in each block to all PEs in first column of block (Fig 10.3b) using 2D mesh links.



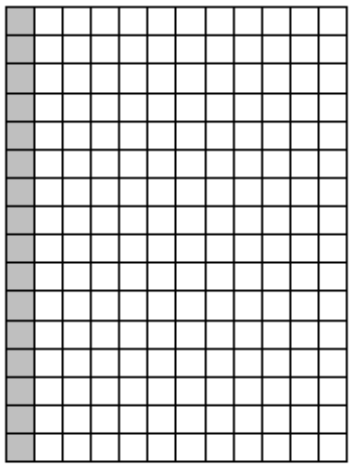
(b)

3. The Y/m partial maxima in each band are divided into m groups of Y/m^2 elements. Each of the m rows in a band are assigned one of these group of Y/m^2 elements (Fig 10.3c). No movement occurs in this step.



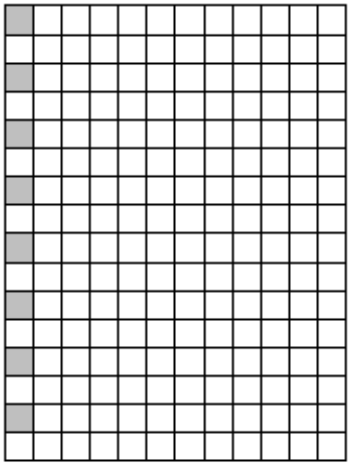
(c)

4. Rows successively broadcast each partial maximum. The leftmost PE in each row computes and stores the maximum of these Y/m^2 values (Fig 10.3d).



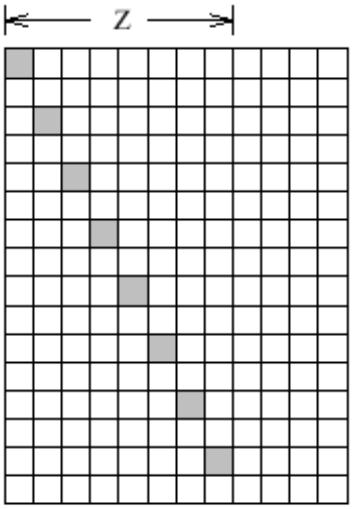
(d)

5. Find the largest of the m partial maxima remaining in each band using second phase of Mesh Maximum algorithm and store in upper left PE (Fig 10.4a).



(a)

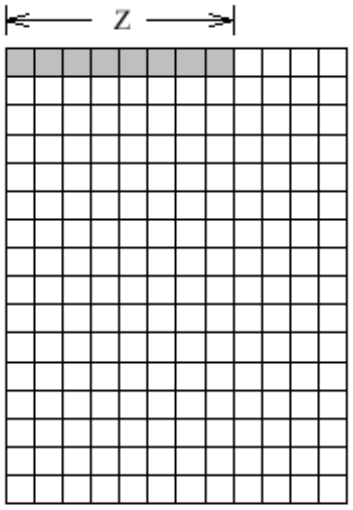
6. The partial maxima in each band j is moved to column $j \bmod Y$ using row broadcasts (Fig 10.4b).



(b)

7. Find the largest of the [at most $X/(Ym)$] partial maxima in each column in Fig10.4b and store it in the top processor (Fig 10.4c).

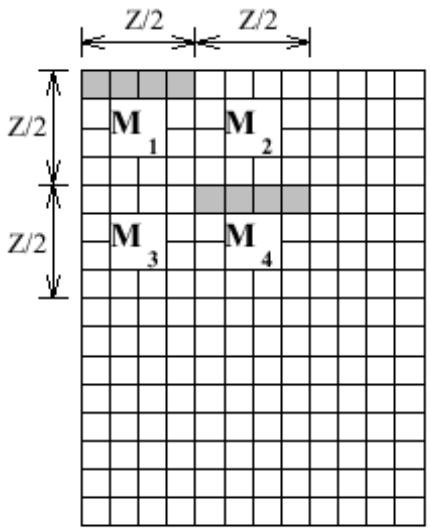
- Partial maxima are successively broadcast along each column and top PE stores largest.
- This reduces the number of partial maxima values to $Z = \min\{Y, X/m\}$



(c)

8. The largest of partial maxima is found recursively (Fig 10.4d).

- Recursively divide remaining problems into two independent subproblems



(d)

- Divide the upper left-hand $Z \times Z$ mesh into four $\frac{Z}{2} \times \frac{Z}{2}$ meshes M_1, M_2, M_3, M_4
- Values are moved from M_2 to M_4 using column buses.
- The set of rows (respectively, columns) of M_1 and M_4 are disjoint.
- Recursion division continues until 1×1 meshes are formed.
- Results from two submesh pairs are merged as follows:
  - Let $m_1$in M_1 and m_4 in M_4 be submesh maximal

values stored in upper left PE of each submesh.

- m_4 is sent to the row containing m_1 using a column bus
- m_4 is sent to the PE containing $m_1$using a row bus.
- The PE in upper left corner the first submesh computes the maximal value for the larger (i.e., parent) mesh containing the two submeshes.

  - This recursion allows maxima values of pairs to be calculated in parallel using recursive doubling

- As argued in Akl's textbook, the running time is minimized when

$$m = n^{1/8},\ X = n^{5/8},\ Y = n^{3/8}$$

- In this case, the running time is

$$t(n) = n^{1/8}$$

which is considerably faster than the $O(\sqrt[3]{n})$ time obtained for the Global Bus Mesh Maximum Algorithm earlier in Chapter 10 of Akl's textbook.

The Reconfigurable Mesh (RM)

- The Reconfigurable Mesh consists of a 2D mesh, augmented with reconfigurable buses. The reconfigurable buses will be discussed below.
- The four NEWS ports of a Mesh Processor (Fig. 10.5):

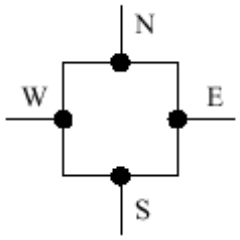


Figure 10.5: The four ports of a mesh processor.

- Possible internal configurations of processor ports (Fig. 10.6):

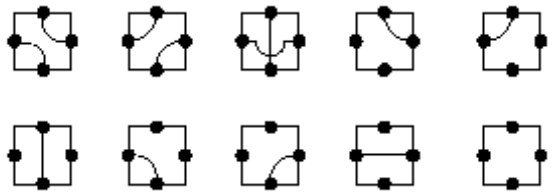


Figure 10.6: Possible internal connections of a processor's ports.

- Each processor can connect *zero or more disjoint pairs* of ports.

- Bus Path Properties:
  - Multiple paths created and changed dynamically, as needed, during the execution of an algorithm.
  - The exact port connections made by a PE at a given step in the algorithm can depend upon their location within the mesh or upon a value in some register.
  - All port connections can be made in O(1) time, allowing multiple bus paths to be created in one step of the algorithm.
  - A single bus can be created that joins all PE, so the algorithms of the Mesh with a Global Bus can be supported.
  - A row bus for each mesh row can be

created. Likewise, a column bus for each mesh column can be created. This allows the RM to support all of the MMB algorithms.

- Also, row buses and column buses can be supported simultanously.
- Many multiple simultanous buses are possible (Fig. 10.7):

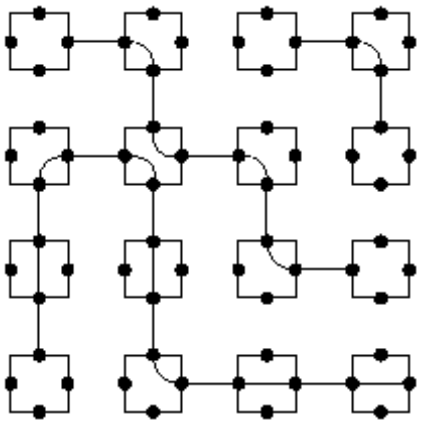


Figure 10.7: A mesh with three configured buses.

- ***A Reconfigurable Mesh Sort : Setup***
  - Consider a mesh with n rows and n^2 columns.
  - We view this mesh as n meshes of size $n \times n$, numbered from 0 to $n - 1$, placed side by side

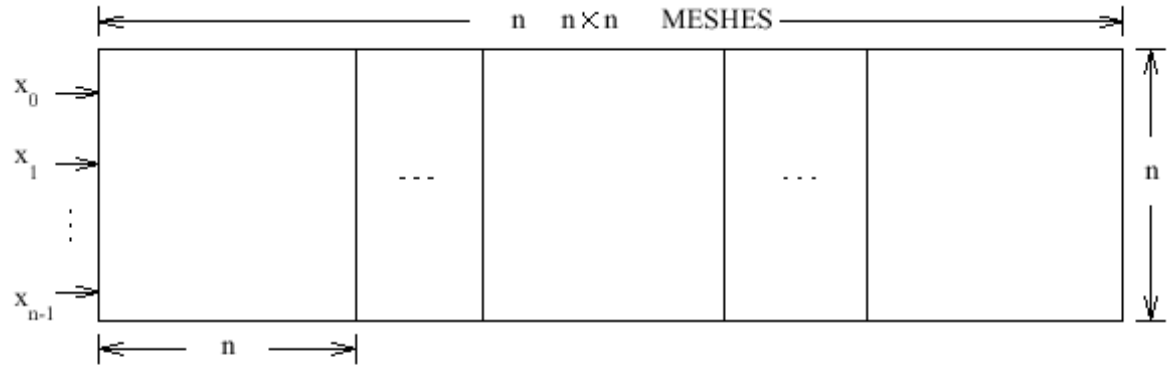


Figure 10.8: A mesh of meshes with reconfigurable buses.

- The numbers, $Q = \{x_0, x_1, \ldots, x_{n-1}\}$ are fed into the left hand column.
- The technique used is called *sorting by enumeration.*
- Each number is compared to each of the other numbers to determine its *rank.*
  - If two numbers are equal, the one with the smaller index is considered to be the smaller in determining its rank.

- ***Reconfigurable Buses Mesh Sort(Q)***

  Step 1 Distribution

  - **1.1** All PEs connect their W and E ports, creating a bus on each

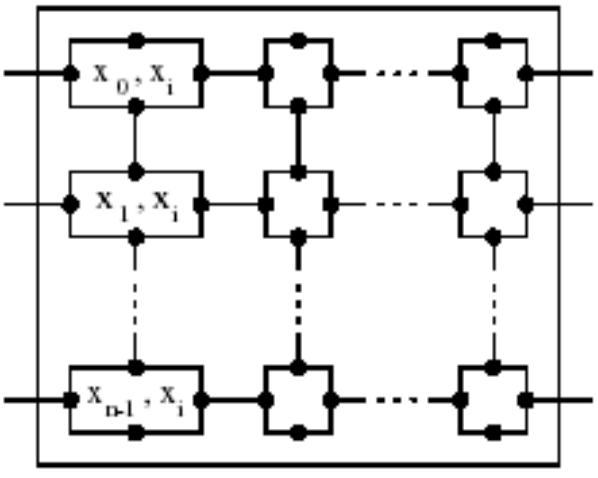
row of the mesh.

1.2 Each $P(i,0)$ of mesh 0 broadcasts x_i to all PEs on row i.

1.3 The PEs in column 0 of $n \times n$ meshes connect their N and S ports, creating a bus on that column.

1.4 Each processor $P(i,0)$ in mesh i broadcasts x_i on the column bus. As a result, in mesh i, $P(j,0)$ contains both x_i and x_j.

- See Fig. 10.10(a)



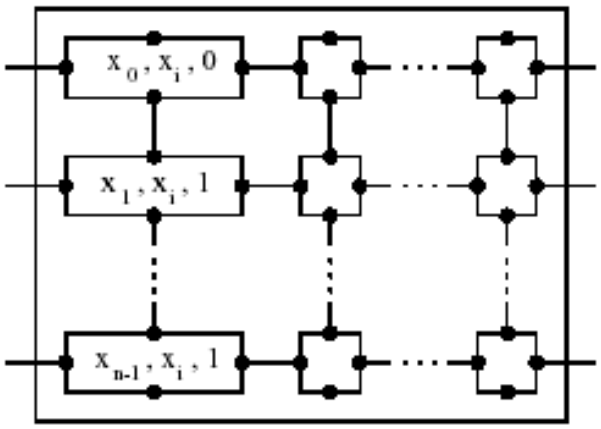
(a)

Step 2 Comparison

- **2.1** In mesh i, processor $P(j,0)$ compares x_i and x_j.

  2.2 If $x_j < x_i$, it stores a 1 in its register R. Otherwise, a 0 is stored in R.

  - See Fig 10.10(b)



(b)

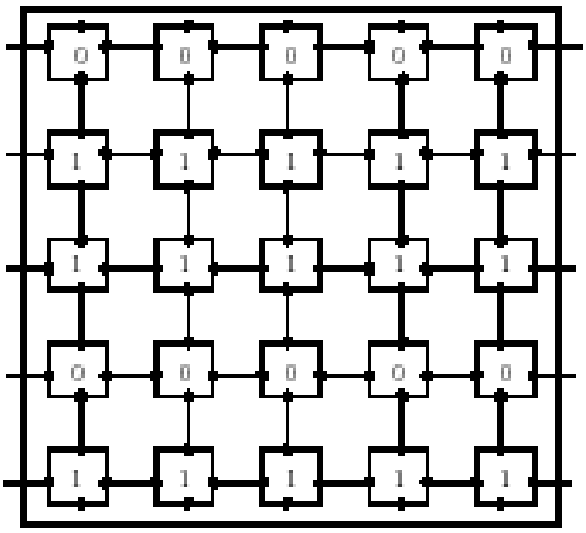
Step 3 Ranking and Enumeration: The following steps are executed by **all** meshes.

- **3.1** All PEs in columns 1 to $n-1$ connect their W and E port in all meshes. This creates a bus on each row.

  3.2 $P(j,0)$ broadcasts the $0/1$ value in its R register on the

bus in row j. All PEs in row j store this value in their register.

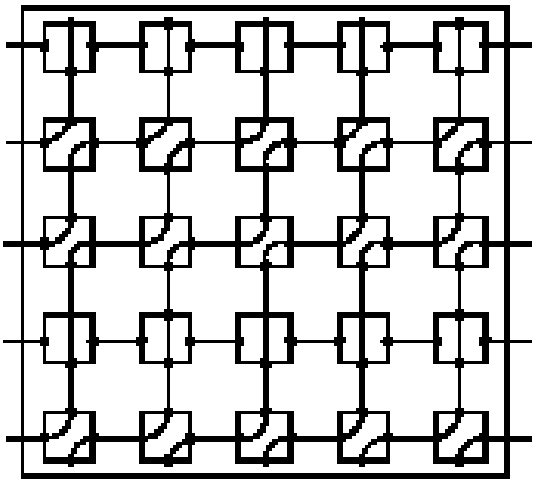
- See Fig 10.11(a) for contents in R register



(a)

3.3 If a PE contains a 0 in its R register, it connects its N and S ports; otherwise, it connects its W and N ports and its S and E ports.

- See buses created in Fig 10.11(b).

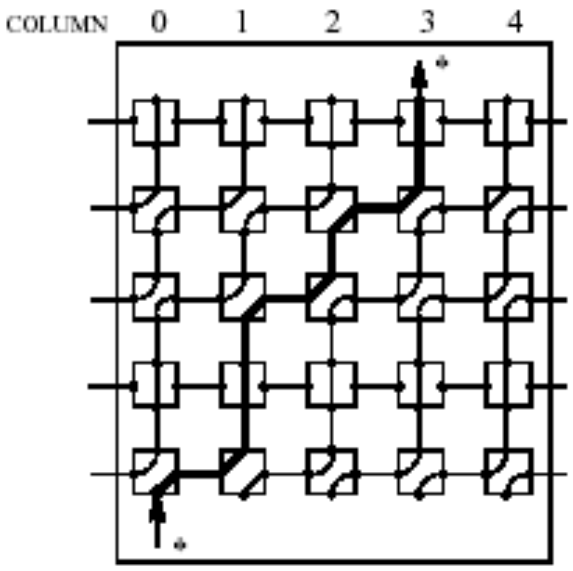


(b)

3.4 The bottom-left PE, $P(n-1,0)$, places a special symbol (say "$*$") on the bus connected to its S port.

3.5 One $P(0,j)$ in the top row of mesh i will receive the $*$. The value j is the rank of i.

- See Fig 10.11(c). j is nr of 1's in a column.



(c)

Step 4 Permutation: The element whose rank is k is output on row k.

- **4.1** In each mesh i, each PE connects its N and S ports, creating a column bus.

  4.2 The processor $P(0,j)$ in the first row receiving a $*$ broadcasts the rank j of x_i down its column bus

  - See Fig 10.12(a).

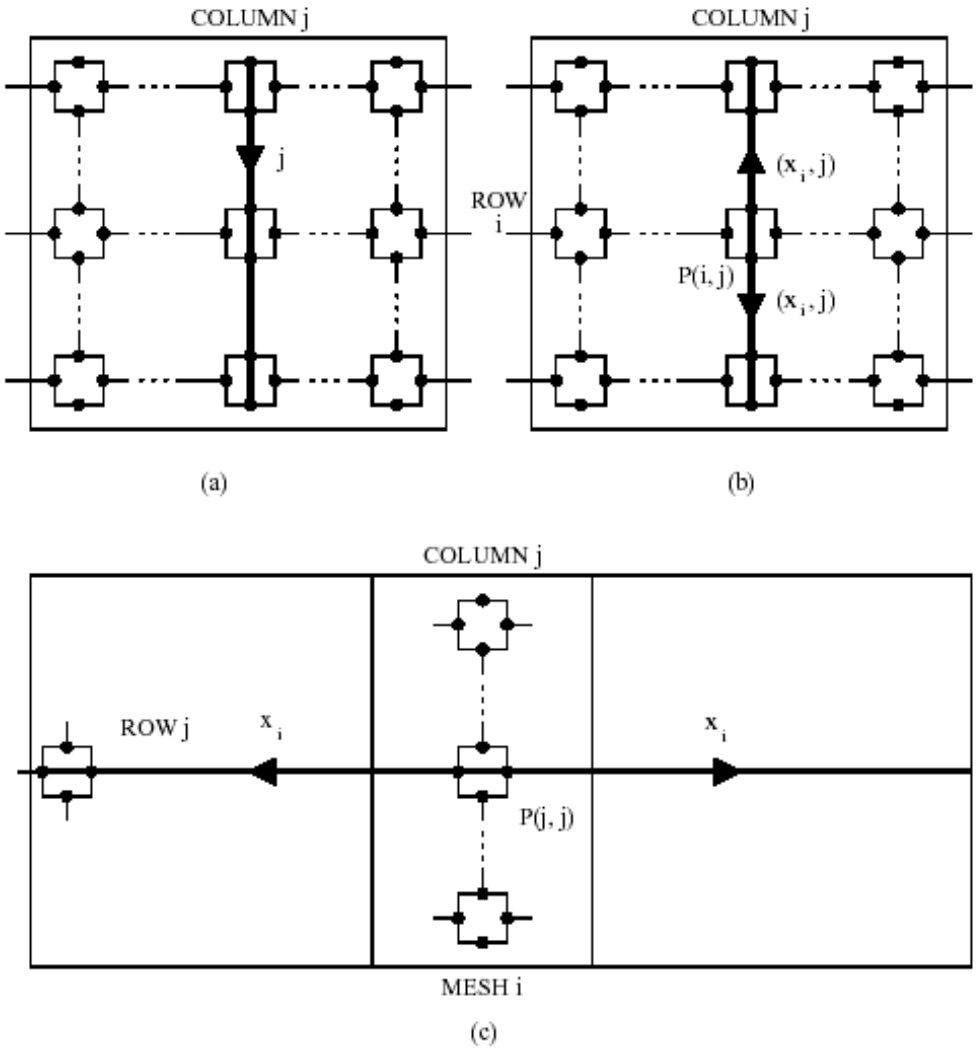
  4.3 $P(i,j)$ now contains both x_i (from step 1) and j. It broadcasts (x_i,j) along column j.

- See Fig 10.12(b).

4.4 All processors of the $n \times n^2$ mesh connect their W and E bus, creating a row bus across the entire mesh.

4.5 $P(j,j)$ of mesh i broadcasts x_i along its row bus.

- Figure 10.12



- ***Analysis of Algorithm***
  - Each step of the algorithm runs in constant time.
  - Therefore,

$$t(n) = O(1)$$

$$p(n) = n^3$$

$$c(n) = O(n^3)$$

  - Some of the techniques used here are quite unique, including the technique used in Step 3 to compute the sum of n bits in $O(1)$ time.
  - It demonstrates the power of the reconfigurable mesh. No interconnection network model studied previously has a ”Constant Time Sorting” algorithm. (Actually, Combining CRCW PRAM can sort in constant time. (e.g., prob. 8.27)
  - The use of processors is exhorbant, but will be reduced in the next algorithm.

- **Problem 10.0**
  - **Part (a):** Show that any permutation $(p_0, p_1, \ldots, p_{n-1})$ of $(x_0, x_1, \ldots, x_{n-1}\}$ can be obtained in constant time using the preceding $O(n^3)$ Reconfigurable Mesh.
  - **Part (b):** The values in the first row or column of an $n \times n$ reconfigurable mesh can be cyclically shifted in constant time.
- The preceding problem is assumed in the next algorithm.

- ***Preliminaries: A More Efficient RM Sort***
  - The basis for this algorithm will be the Mesh Sort.
  - Mesh Sort only uses 3 basic operations
    - Sorting a row of a matrix
    - Sorting a Column of a matrix
    - Cyclic shifting a row of a matrix.
  - Assume that the values $Q = \{x_0, x_1, \ldots, x_{n-1}\}$ are organized

into an array $A = X \times Y$ where

$$X = n^{2/3} \text{ and } Y = n^{1/3}$$

- We associate each row of A with Y^3 processors, organized as a sequence of Y meshs of size $Y \times Y$
  - Equivalently, one mesh with Y rows and Y^2 columns.
  - Lets visualize these as attached to rows below A and $\perp$ to plane containing A.
  - See Figure 10.13 in Akl's textbook.
- We associate each column of A with X^3 processors, organized as a sequence of X meshes of size $X \times X$.
  - Equivalently, a mesh of X rows and X^2 columns.
  - Lets visualize these $X \times X^2$ meshes attached to each column in A and in a plane $\perp$ to plane containing A and lying above the column of A they attach to.

- See Figure 10.13 in Akl's textbook.

- The total number of PEs required are

$$\begin{aligned} p(n) &= X \times Y^3 + Y \times X^3 - X \times Y \\ &= n^{\frac{2}{3}} n + n^{\frac{1}{3}} n^{\frac{6}{3}} - n^{\frac{2}{3}} n^{\frac{1}{3}} \\ &= n^{5/3} + n^{7/3} - n \\ &= O(n^{7/3}) \subset O(n^{2.34}) \end{aligned}$$

  which is much better than $O(n^3)$

- ***Algorithm: A More Efficient RM Sort***
  1. Whenever **Mesh Sort** calls for a row of A to be sorted, the Y^3 attached PEs execute **Reconfigurable Buses Mesh Sort.**
  2. Whenever **Mesh Sort** calls for a column to be sorted, the attached X^3 PEs execute **Reconfigurable Buses Mesh Sort.**
  3. Whenever **Mesh Sort** calls for a row to be cyclically shifted, the attached Reconfigurable Mesh is

used to produce this shift in $O(1)$ time.

- *Problem* 10.0, which was included earlier, justifies this.

4. Whenever cyclic shifts of rows within all vertical strips is required, this is just a permutation of each row and is handled in the same way as (3) above.

- ***Algorithm Analysis***
  - We have already shown that
    $$p(n) = O(n^{7/3})$$
  - The running time is
    $$t(n) = O(1)$$
    since each step in **Mesh Sort** can be executed in constant time.
  - The cost,
    $$c(n) = O(n^{7/3})$$
    is an improvement over the $O(n^3)$ cost of the previous algorithm.
- ***A 3D Reconfigurable Mesh***

Improvement

- Note that the 3*D* space above *A* is large enough to contain the 3*D* space below *A*.
- The two spaces can not be combined without adding 3*D* mesh and bus connections for at least the first $X \times Y^2$ block of PEs in the X^3 block above *A*.
- If a 3*D* Reconfigurable Mesh connections are allowed, a $\sqrt{n} \times \sqrt{n} \times \sqrt{n}$ cube can support a constant time sort of *n* items. Additionally, only the base plane has to contain PEs and the rest can be switches (i.e, components).
- This reduces the cost of the above algorithm to $c(t) = O(n^{3/2}),$ even if PEs are used instead of switches.
- Reference: ”A Constant Time Sorting Algorithm for a 3*D* Reconfigurable Mesh and Reconfigurable Network”, M. Merry and J. Baker, Parallel Processing Letters, vol 5, 1995,

401-412.

Optical Buses

- Models of Computation
  - Usefulness depends upon their ability to capture true computing engines.
  - Types of assumptions:
    - Based on what is *currently feasible* and what is expected in the *foreseeable future.*
    - Includes important aspects of their computation.
    - Ignores aspects of computation that are of secondary importance
    - Includes time for operations and travel along the network.
    - Important that models allows the general performance of algorithms to be predicted and compared.
    - Additionally, some models allow the running time of algorithms to

be accurately estimated.
    - Based on the data size and execution time for various basic operations.
- Two Properties of Electronic Buses:
    - bidirectionality:
        - Datum placed on a bus by a processor *P* travels in both directions away from *P*.
    - Speed of electronic signals on a bus:
        - There is no precise function to compute the speed a signal travels along a bus.
        - Customary to assume that the speed is infinite and arrives at all PEs on the bus instantaneously.
- Important Optical Bus Properties:
    - Unidirectional:
        - Datum placed on a bus travels in one direction.
    - Propogation Delay:
        - Predictable

        - distance traveled is directly proportional to time.
- Use of Optical Buses
    - Previous assumptions support forming a *pipeline.*
    - If P_1 and P_2 put their datum on the bus at the same time, the difference in the times these two datum arrive at a third processor P is predictable.
    - See Figure 10.14 in Akl's textbook

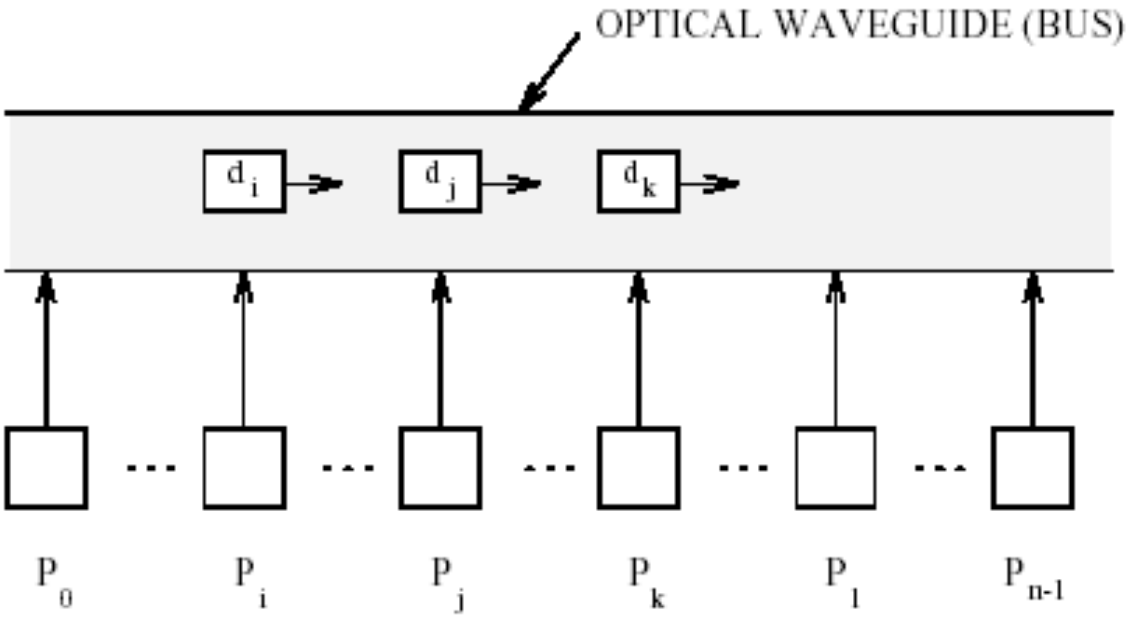


Figure 10.14: A pipeline of data on an optical bus.

- **Linear Arrays with Optical Buses**

- Let $P_0, P_1, \ldots, P_{n-1}$ be processors connected by a two-way link to an optical bus.
- One edge is for receiving data and the other for sending data.
- Data travels on the bus in one direction only.
- Each datum placed on the bus consists of b bits.
- Two successive PEs are a fixed number D of light waves apart.
- The number of time units required for a light pulse to traverse D is denoted τ_D, where

  $$\tau_D = D/v$$

  and v is the speed of light in the waveguide.
- If $j > i$ and P_i sends P_j a message, then it arrives after (j-i)τ_D time units.
- Multiple PEs can place their datum on a bus simultaneously.
- See Figures 10.14 and 10.15 in Akl's

textbook.

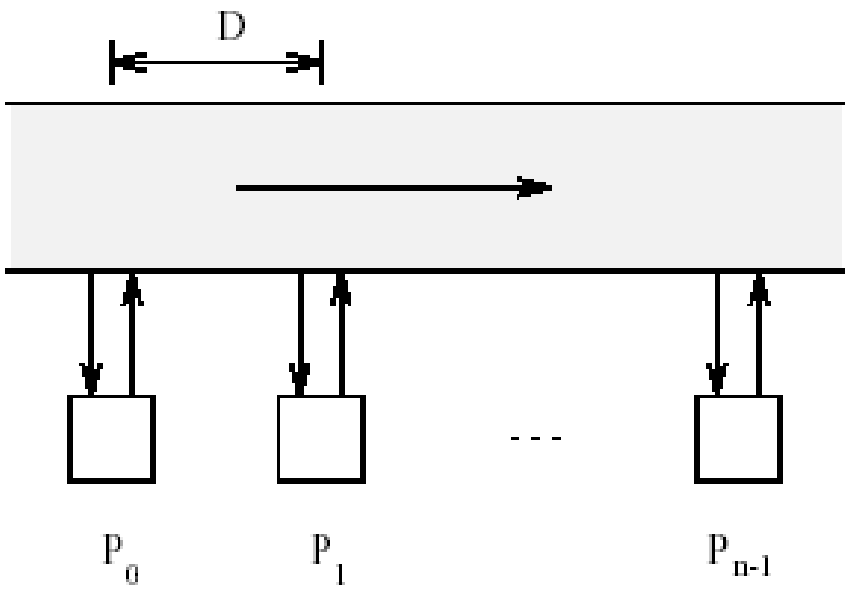


Figure 10.15: A linear array of processors with an optical

- A bit is represented by a light pulse of w time units duration.
- In order to avoid overlapping messages, the following conditions must be satisfied:
  - $D > bwv$
  - PEs must write to the bus at pre-specified times, separated by regular time intervals.
- A *bus cycle* is the time $\tau_{B(L)}$ for an optical signal to traverse the bus from

one end to the other.

- Since the optical length of the bus is $L = (n-1)D$ and

$$\tau_{B(L)} = L/v,$$

we assume $\tau_{B(L)}$ is $O(1)$.

- **The Wait Function**
  - **Case I (Receive)** Each receiving processor P_j knows the identity of the sending processor P_i.
    - All PEs wishing to send a message place a datum on the bus at the beginning of the bus cycle.
    - We assume that all PEs write, with the ones without messages sending a dummy message.
    - P_j skips $j - i - 1$ messages and reads the $(j-i)$th message that passes by.
    - The function

$$wait(i,j) = (j-i)\tau_D$$

specifies the time that that P_j must wait before reading datum d_i.

- Since τ_D is a constant, we simplify the notation here by assuming it is 1 and simply writing

$$wait(i,j) = j - i$$

- In one bus cycle,
  - The same message can be read by many PEs
  - Each PE can read only one message
  - In Akl's textbook, see Figure 10.16

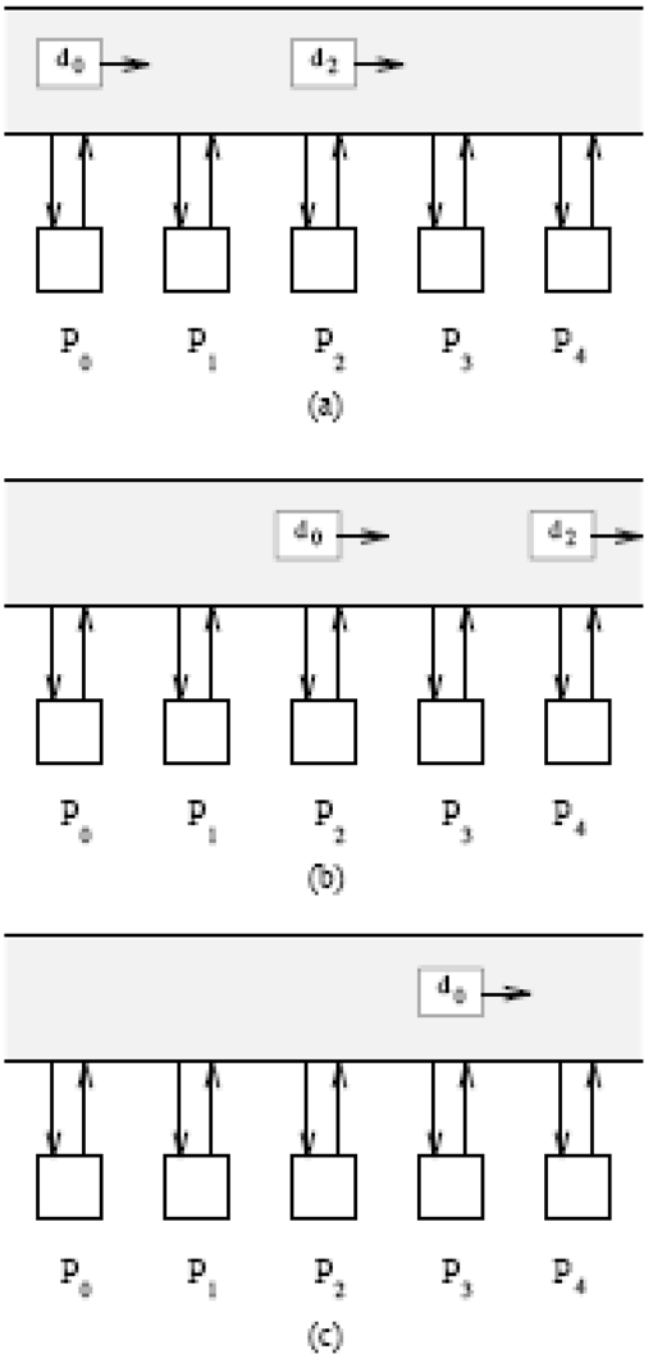


Figure 10.16: Routing data when receiver "knows" sender: (a) At the beginning of the bus cycle, P_0 places d_0 (destined to P_3) on the bus, while P_2 places d_2 (destined to P_4) on the bus; (b) P_4 receives d_2 two time units after the beginning of the bus cycle; (c) P_3 receives d_0 three time units after the beginning of the bus cycle.

- **Case II (Send)** The receiver P_j does not know the identity of the sender P_i but the sender P_i knows the identity of the receiver P_j.
  - Sender P_i writes its message d_i on the bus at time

    $$(n - 1) - (j - i)$$

    relative to the bus cycle.
  - All PEs simultaneously read the bus at the end of the bus cycle.
  - If there is a message for one or more PEs, the receiver will find it there at that time.
  - In Akl's textbook, see Figure 10.17

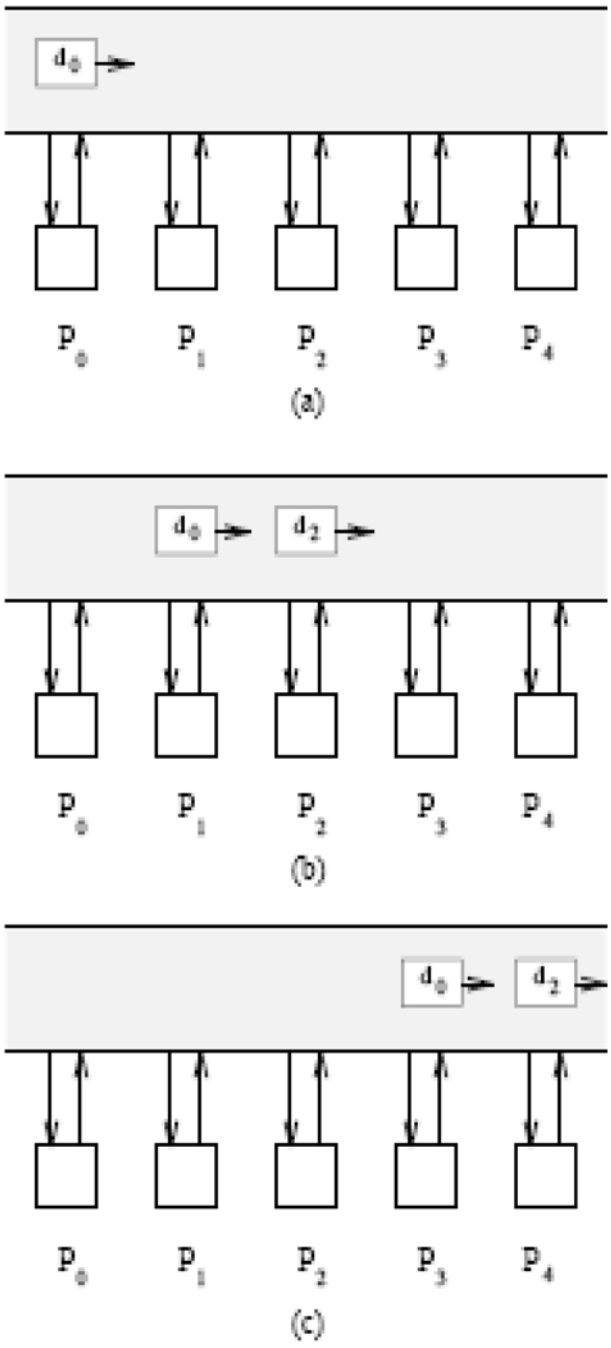


Figure 10.17: Routing data when sender "knows" receiver: (a) P_0 places d_0 (destined to P_3) on the bus one time unit after the beginning of the bus cycle; (b) P_2 places d_2 (destined to P_4) on the bus two time units after the beginning of the bus cycle; (c) Each datum reaches its destination at the end of the bus cycle.

- **Supporting Two Way Communications**
  - The system in Figures 10.14 &10.15 allow travel in only one direction.
  - Data movement in both directions is supported using two optical buses (see Fig. 10.18).

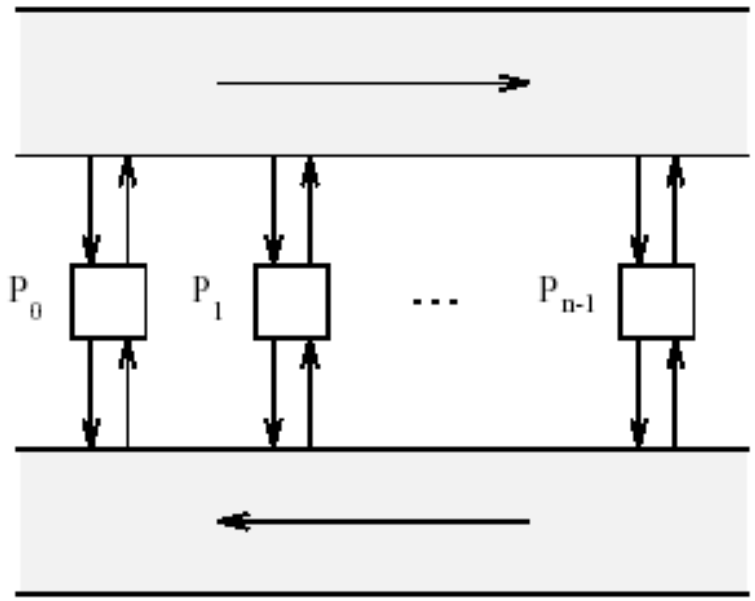


Figure 10.18: Two-way communication using optical buses.

- Data travels left to right on one bus and travels right to left along the second bus.
- Each PE can read and write to either of the two buses.

- The two buses are synchronous and support separate pipeline.
- The definition of the *wait* function is extended as follows: For $i \neq j$, if $wait(i,j) > 0$ then P_j reads from the left-to-right bus, otherwise it reads from the right-to-left bus.

- Using the *wait* function, any communication pattern (e.g., broadcasting a datum from one PE to all others, executing an arbitrary permutation of the data, reductions, etc.) can be specified.
- **Data Communications** (assuming 2-way communications)
  - **Broadcast:** If P_i is to broadcast to all other processors , for each P_j with $j \neq i$ define

$$wai(i,j) = j - i$$

    - The entire broadcast operation requires one bus cycle.

- This is a *'receive'* operation

- **Permutations:** Suppose an arbitrary permutation r is required, so that

$$d_i \rightarrow P_{r(i)}$$

and each processor receives exactly one datum. It suffices to set

$$wait(i, r(i) = r(i) - i$$

for all i.

  - The entire permutation is completed in one bus cycle.
  - A permutation can be handled by either a 'send' or a 'receive' operation.

- **Data Mappings:** Let $f(j)$ specifies the the data that j will receive. Then

$$f : \{0, 1, , , , , n-1\} \rightarrow \{0, 1, 2, \ldots, n-1\}$$

and we want P_j to receive d_j from $P_{f(j)}$

- While f is a function, it may fail to be a permutation (which also requires f to be 1-1 and onto). In particular, there could be values j and k where $0 \leq j, k < n$ and $j \neq k$ but

$$f(j) = f(k) = i$$

- In all cases, the *wait* function is defined by

$$wait(f(j), j) = j - f(j)$$

- Note in the preceding formula for *wait*, that $f(j)$ is the location of the data sent to P_j.
- More than one processor can receive the same data item f_j as $f_j = f_k$ for j ≠ k is possible]
- This requires a ’receive’ operation since more than one PE can receive the same datum.
- Note that a permutation is a

special case of this operation.

- A consequence of the preceding is that a linear array with optical buses can simulate PRAM in constant time, as summarized in the next theorem.
- **PRAM Simulation Theorem:** A linear array with optical buses (LAOB) and n PEs and $O(1)$ memory locations per PE can simulate CREW PRAM with n PEs and $O(n)$ shared memory locations in constant time.
  - The LAOB simulating model will have the same number of PEs as the PRAM model being simulated.
    - The PEs used in the two models will be assumed to be identical. so that they will have the same capabilities.
  - All three of the data communication operations, i.e., broadcasting, permutation, and data distribution can be performed

within one bus cycle.

- That is, all three of these operations can be done in constant time since $\tau_{B(L)}$ is assumed to take constant time (i.e., no more time than a basic operation such as comparing two numbers).

- The **ER** and **EW** PRAM communication operations are permutations, if one allows some data values to be ”null values”.
- Also, a **CR** can be viewed as a data distribution operation, again if some data values are allowed to be ”null”.

- The PRAM **CW** operation can not be simulated in constant time by LAOB.
  - Since a **CR** involves having some PEs receive an arbitrary number of values in one step, this can not be accomplished in one LAOB step.

        - A **CR** can be viewed as an arbitrary number of **ER** steps.
- **Meshes with Optical Buses**
    - Two problems with optical buses
        - Optical Signals weaken rapidly as they travel long distances.
        - The time for a message to travel the length of the bus grows linearly with the length of the bus.
    - When the number of PEs are large, the bus length can be decreased by placing the PEs in a $\sqrt{n} \times \sqrt{n}$ 2D mesh pattern (see Figure 10.21 in Akl's textbook).

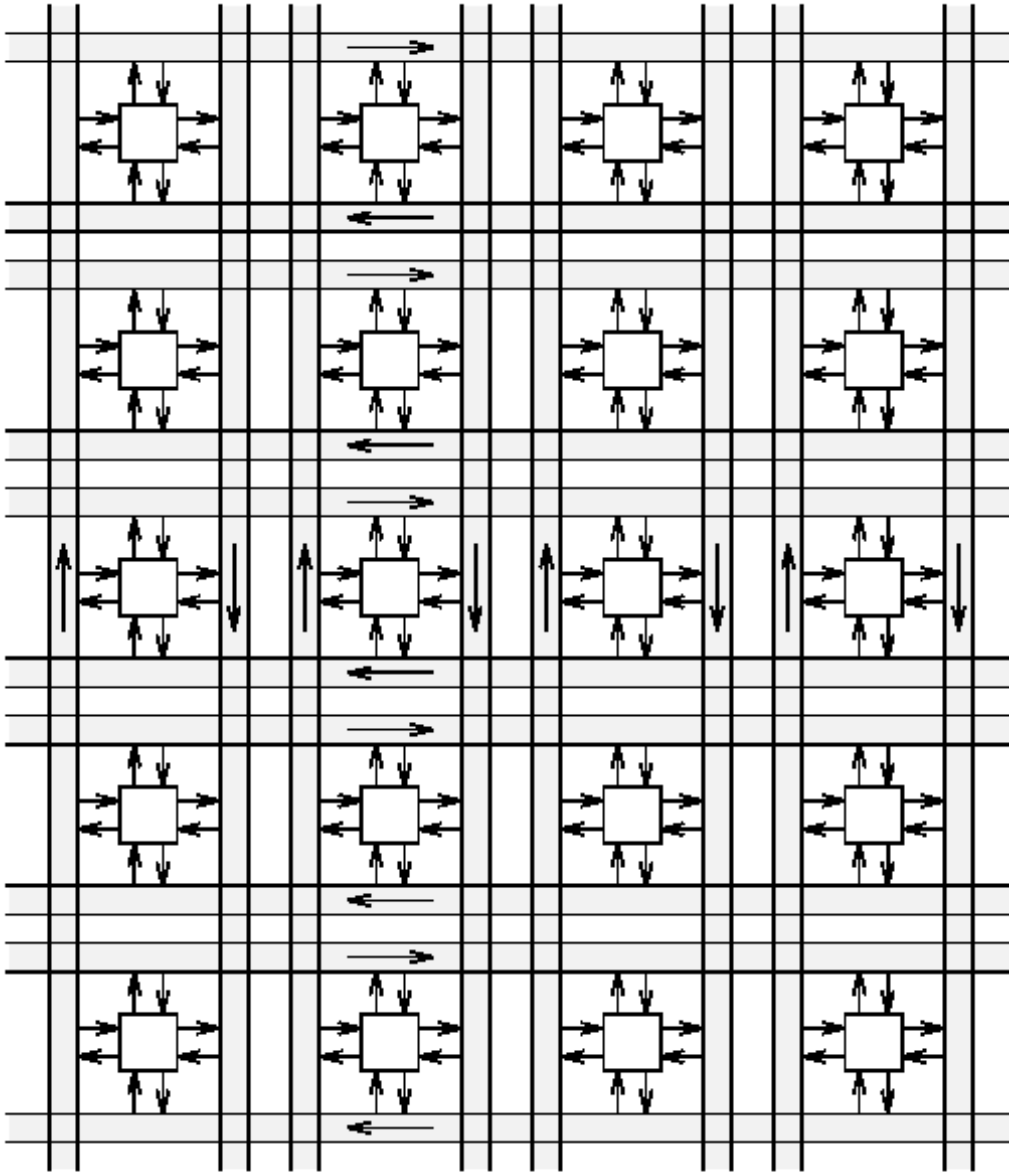


Figure 10.21: A mesh of processors with optical buses.

- Observations:
  - No two PEs are joined by standard 2D mesh links. Only buses are used to move data.

- A message can be sent between any two PEs in two bus cycles.

- A Sorting Algorithm
  - **Claim:** The *Mesh Sort* of Chapter 8 can be executed by a Mesh with Optical Buses (MOB) in $O(lgn)$ time.
  - Each step of Mesh Sort executes one or more of the following operations:
    - Sort a row
    - Sort a column
    - Perform a cyclic shift within rows.

      Comment: A row permutation is more general than a cyclic shift of a row.
  - Suppose the Mesh with Optical Buses consists of $X = 2^s$ rows and $Y = 2^{2r}$ columns where $s \geq r$ and $XY = n$.

    1. Whenever a row (or

column) is to be sorted, the PEs in that row (column) will simulate PRAM SORT.

2. Whenever a row is to be cyclically shifted, this is done using the *wait* function since this is just a permutation.

- *Algorithm Analysis*: Requires $O(\lg n)$ time and n PEs for an optimal cost of $O(n \lg n)$. A more detailed analysis follows:
  - Since CW is not needed in this sort, Algorithm PRAM SORT (see Akl, pg 179) can be used.
    - PRAM SORT can be simulated in constant time by a linear array with optical buses.
    - **Comment:** The PRAM SORT was discussed in the PRAM chapter and is the Cole Sort, which is valid for

EREW (Also, see reference 166 in Akl's textbook).

- For both rows and columns, PRAM sort will sort n^t numbers using n^t PEs in $O(\lg n^t) = O(\lg n)$ time for some t with $0 < t < 1$.
  - Recall $X = 2^s$ rows and $Y = 2^{2r}$ columns where $s \geq r$ and $XY = n$.
- There are 13 sorting steps (9 column and 4 row sorts).
- Each permutation requires one bus cycle and there are 4 cycles/permutations in Mesh Sort.
- Consequently, a sequence of length n can be sorted in

$$t(n) = O(\lg n)$$

using $p(n) = n$ PEs.

- The above sort is cost optimal since

$$c(n) = O(n \lg n)$$