Linear Arrays

Chapter 7

1. Basics for the linear array computational model.

a. A diagram for this model is

$$P_1 \leftrightarrow P_2 \leftrightarrow P_3 \leftrightarrow \ldots \leftrightarrow P_k$$

b. It is the simplest of all models that allow some form of communication between PEs.

c. Each processor only communicates with its right or left neighbor.

d. We assume that the two-way links between adjacent PEs can transmit a constant nr of items (e.g., a word) in constant time

e. Algorithms derived for the linear array are very useful, as they can

can be implemented with the same running time on most other models.

f. Due to the simplicity of the linear array, a copy with the same number of nodes can be embedded into the meshes, hypercube, and most other interconnection networks.
- This allows its algorithms to executed in same running time by these models.
- The linear array is weaker than these models.

g. PRAM can simulate this model (and all other fixed interconnection networks) in unit time (using shared memory).
- PRAM is a more powerful model than this model and other fixed interconnection network models.

h. Model is very *scalable*: If one can build a linear array with a certain clock frequency, then one can also build a very long linear array with the same clock frequency.

i. We assume that the two-way link between two adjacent processors has enough bandwidth to allow a constant number of data transfers between two processors simultaneously

- E.g., P_i can send two values a and b to P_{i+1} and simultaneously receive two values d and e from P_{i+1}
- We represent this by drawing multiple one-way links between processors.

2. Sorting assumptions:

a. Let $S = \{s_1, s_2, \ldots, s_n\}$ be a sequence of numbers.

b. The elements of S are not all available at once, but arrive one at a time from some input device.

c. They have to be sorted ”on the fly” as they arrive

d. This places a lower bound of $\Omega(n)$ on the running time.

3. **Linear Array Comparison-Exchange Sort**

a. **Figure 7.1** illustrates this algorithm:

$$\begin{array}{ll} \ldots s_3 s_2 s_1 & \rightleftarrows P_1 \rightleftarrows P_2 \rightleftarrows \ldots \rightleftarrows P_k \\ \textit{output} & \end{array}$$

b. The first phase requires n steps to read one element s_i at a time at P_1.

c. The implementation of this algorithm in the textbook require n PEs but only PEs with odd indices do any compare-exchanges.

d. The implementation given here for this algorithm uses only $k = \lceil n/2 \rceil$ PEs but has storage for two numbers, $upper$ and $lower$.

e. During the first step of the **input**

phase, P_1reads the first element $s_1$into its *upper* variable.

f. During the *jth* step ($j > 1$) of the **input phase**
- Each of the PEs $P_1, P_2, \ldots, P_j$ with two numbers compare them and swaps them if the *upper* is less than the *lower*.
- A PE with only one number moves it into *lower* to wait for another number to arrive.
- The content of all PEs with a value in *upper* are shifted one place to the right and P_1reads the the next input value into its *upper* variable.

g. During the **output phase**,
- Each PE with two numbers compares them and swaps them if if *upper* is less than *lower*.
- A PE with only one number moves it into *lower*.

- The content of all PEs with a value in *lower* are shifted one place to the left, with the value from P_1being output
- numbers in *lower* move right-to-left, while numbers in *upper* remain in place.

h. **Property:** Following the execution of the first (i.e., comparison) step in either phase, the number in *lower* in P_i is the minimum of all numbers in P_j for $j \geq i$ (i.e., in P_i or to the right of P_i).

i. The sorted numbers are output through the *lower* variable in P_1 with smaller numbers first.

j. Algorithm analysis:
   - The running time, $t(n) = O(n)$ is optimal since inputs arrive one at a time.
   - The cost, $c(t) = O(n^2)$ is not optimal as sequential sorting requires $O(n \lg n)$

4. Sorting by Merging

a. Idea is the same as used in PRAM SORT: several merging steps are overlapped and executed in pipeline fashion.

b. Let $n = 2^r$. Then $r = \lg(n)$ merge steps are required to sort a sequence of n nrs.

c. Merging two sorted subsequences of length m produces a sorted subsequence of length $2m$.

d. Assume the input is $S = \{s_1, s_2, \ldots, s_n\}$.

e. **Configuration:** We assume that each PE sends its output to the PE to its right along either an upper or lower line.

input $\rightarrow P_1 \rightrightarrows P_2 \rightrightarrows \ldots \rightrightarrows P_{r+1} \rightarrow$ output

- Note $\lg(n) + 1$ PEs are needed since P_1 does not merge.

f. Algorithm Step j for $P_1$for $1 \leq j \leq n$.

- P_1 receives s_j and sends it to

P_2 on the top line if j is odd and on bottom line otherwise.

g. Algorithm Steps for P_i for $2 \leq i \leq r+1$.

  i. Two sequences of length 2^{i-2} are sent from P_{i-1} to P_i on different lines.

  ii. The two subsequences are merged by P_i into one sequence of length 2^{i-1}.

  iii. Each P_i starts producing output on its top line as soon as it has received top subsequence and first element of the bottom subsequence.

h. **Example:** See **Example 7.2** and (**Figure 7.4** or my expansion of it).

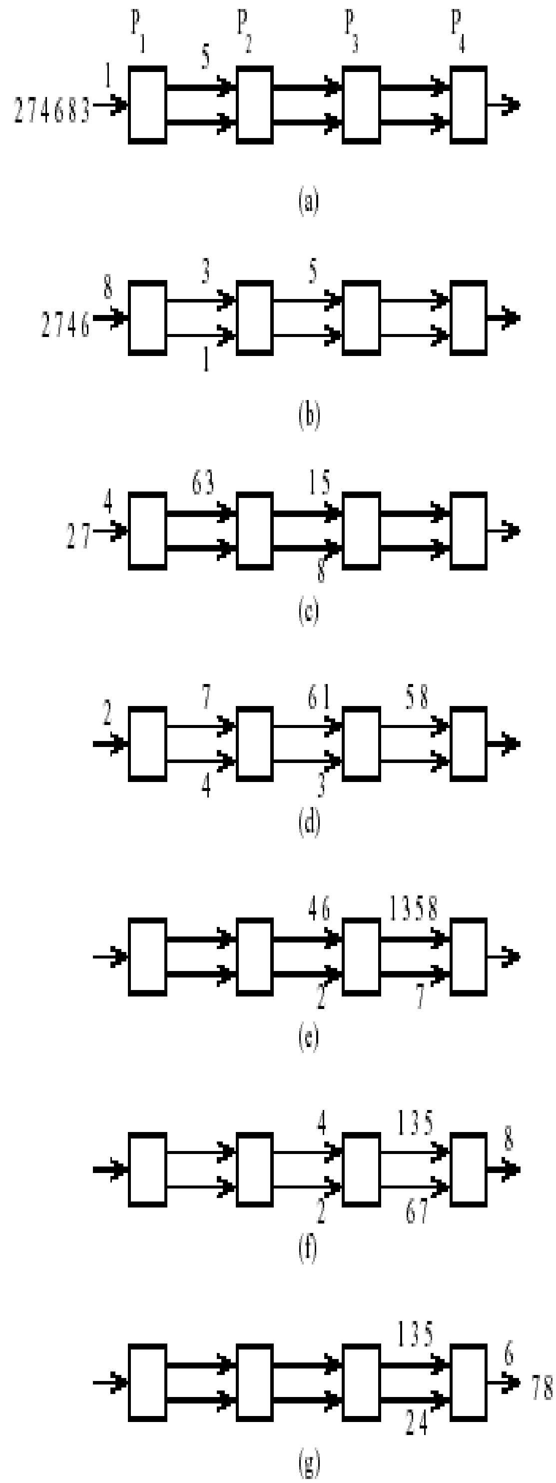

Figure 7.4: Sorting by merging in a pipeline on a linear array: (a) $u = 1$; (b) $u = 3$; (c) $u = 5$; (d) $u = 7$; (e) $u = 10$; (f) $u = 11$; (g) $u = 13$.

i. **Analysis:**

- P_1 produces its first output at time $t = 1$.
- For $i > 1$, P_i requires a subseqence of size 2^{i-2} on top line and another of size 1 on bottom line before merging begins.
- P_i begins operating $2^{i-2} + 1$ time units after P_{i-1} starts, or when

$$t = 1 + (2^0+1) + (2^1+1) + \ldots + (2^{i-2}+1)$$

$$= 2^{i-1} + i - 1$$

- P_i terminates its operation $n - 1$ time units after its first output.
- P_{r+1} terminates last at time

$$t = (2^r + r) + (n - 1)$$

$$= 2n + \lg n - 1$$

- Then $t(n) = O(n)$.
- Since $p(n) = 1 + \lg n$, the cost

is

$$C(n) = O(n \lg n),$$

which is optimal since $\Omega(n \lg n)$ is a lower bound on sorting.

5. Two of H.T.Kung's linear algebra algorithms for special purpose arrays (called *systolic circuits*) are given next.

6. Matrix by vector multiplication:

a. Multiplying an $m \times n$ matrix A by a $n \times 1$ column vector u produces an $m \times 1$ column vector $v = (v_1, v_2, \ldots, v_m)$.

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \end{pmatrix} = \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{pmatrix}$$

b. Recall that

$$v_i = \sum_{j=1}^{n} a_{i,j} u_j \text{ for } 1 \le i \le m$$

c. Processor P_i is used to compute

v_i.

d. Matrix A and vector u are fed to the array of processors (for $m = 4$ and $n = 5$) as indicated in Figure 7.5

e. See **Figure 7.5**

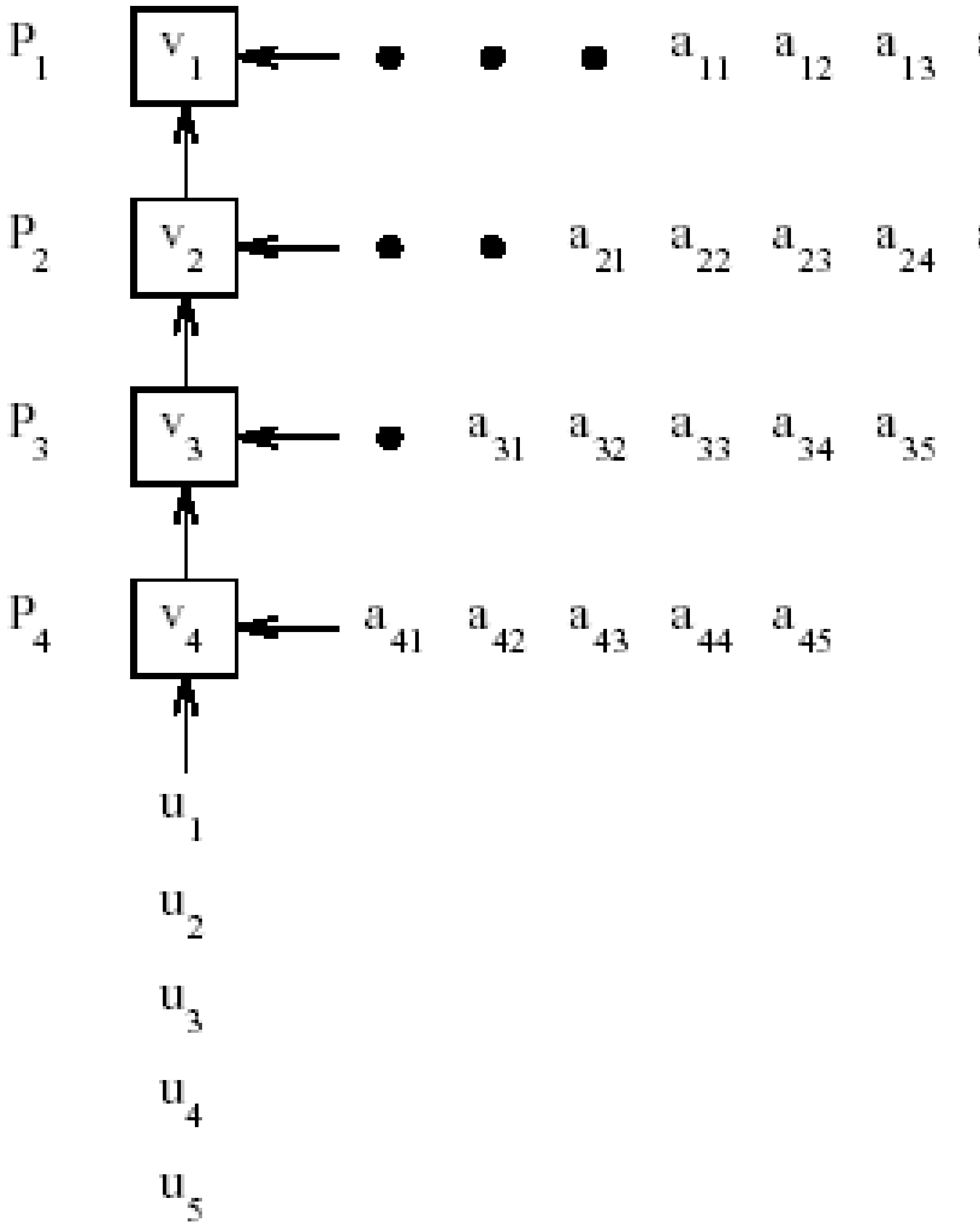

Figure 7.5: Multiplying a matrix by a ve

f. Note that processor P_i computes

$$v_i \leftarrow v_i + a_{ij} u_j$$

and then sends u_j to P_{i-1}.

g. Analysis:

- $a_{1,1}$ reaches P_1 in $m - 1$ steps.
- Total time for $a_{1,n}$ to reach P_1 is $m + n - 2$ steps.
- Computation is finished one step later, or in $m + n - 1$ steps.
- $t(n) = O(n)$ if m is $O(n)$.
- $c(n) = O(n^2)$
- Cost is optimal, since each of the $\Theta(n^2)$ input values must be read and used.

7. **Observation:** Multiplication of an $m \times n$ matrix A by a $n \times p$ matrix B can be handled in either of the following ways:

a. Split the matrix B into p columns and use the linear array of PEs p times (once for each column).

b. Replicate the linear array of PEs p times and simultaneously compute all columns.

8. **Solutions of Triangular Systems** (H.J. Kung)

   a. A *lower triangular matrix* is a square matrix where all entries above the main diagonal are 0.

   b. **Problem:** Given an $n \times n$ lower triangular matrix A and an $n \times 1$ column vector b, find an $n \times 1$ column vector x such that $Ax = b$.

   c. Normal Sequential Solution:

   - *Forward substitution*: Solve the equations

$$\begin{aligned} a_{11}x_1 &= b_1 \\ a_{21}x_1 + a_{22}x_2 &= b_2 \\ \ldots &= \ldots \\ a_{n1}x_1 + \ldots + a_{nn}x_n &= b_n \end{aligned}$$

     successively, substituting all values found for $x_{1,} \ldots, x_{i-1}$ into the i^{th} equation.

   - This yields $x_1 = b_1 / a_{11}$ and, in general,

$$x_i = (b_i - \sum_{j=1}^{i-1} a_{ij}x_j)/a_{ii}$$

- The values for $x_1, x_2, \ldots, x_{i-1}$ are computed successively using this formula, with their values being found first and used in finding the value for x_i.
- This sequential solution runs in $\Theta(n^2)$ time and is optimal since each of the $\Theta(n^2)$ input values must be read and used

d. **Recurrence equation solution to system of equations**: If

$$y_i^{(1)} = 0$$

and, in general,

$$y_i^{(j+1)} = y_i^{(j)} + a_{ij}x_i \text{ for } j < i$$

then

$$x_i = (b_i - y_i^{(i)})/a_{ii}$$

e. Above claim is obvious if one

notes that expanding the recurrence relation for y_i^j (for $j < i$) yields

$$y_i^{(i)} = a_{i1}x_1 + a_{i2}x_2 + \ldots + a_{i,i-1}x_{i-1}$$

f. **EXAMPLE:** See my corrected handout for the following **Figure 7.6 :**

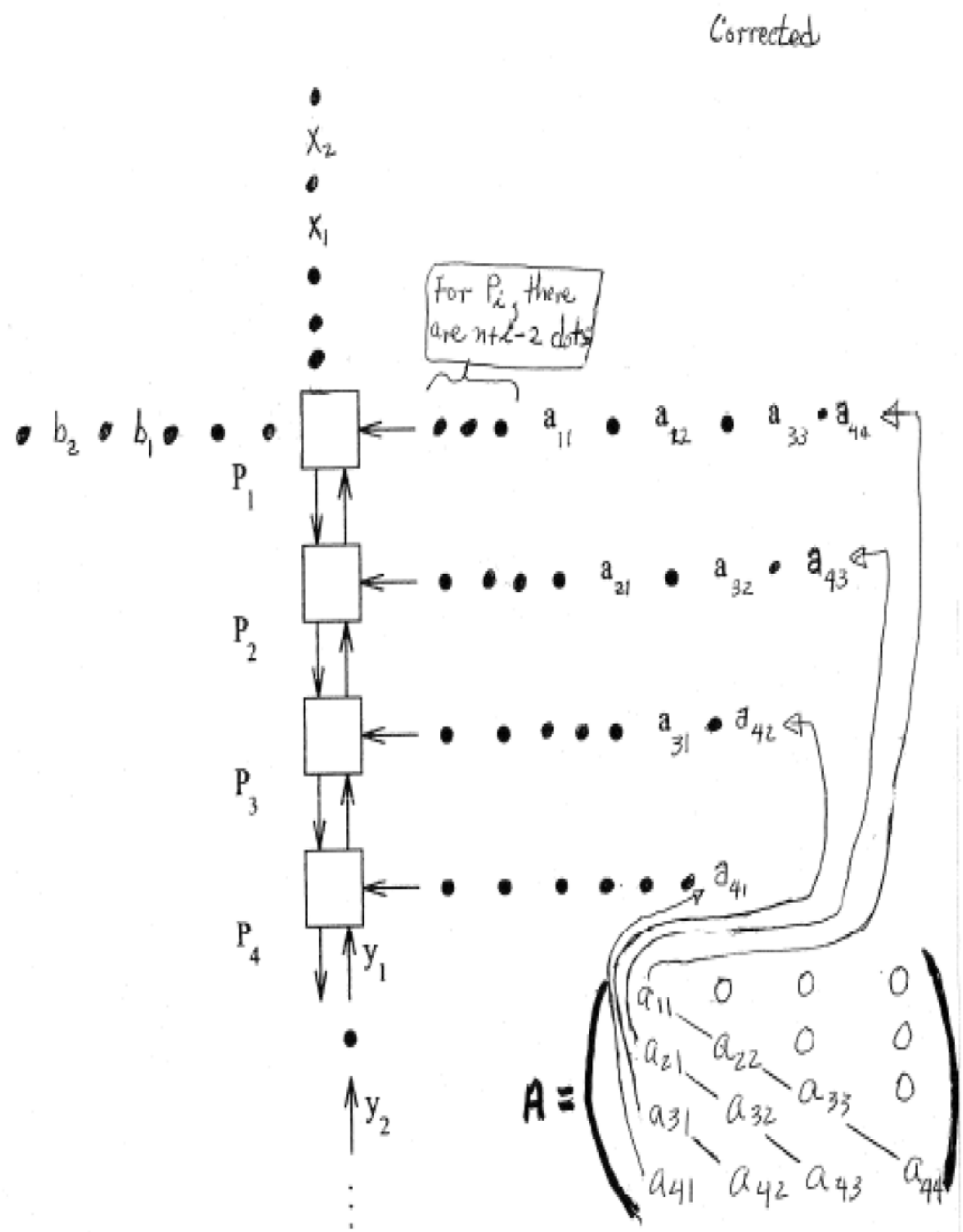

Figure 7.6: Setup for solving a triangular system of equations.

g. Solution given for a triangular system when $n = 4$.

- Example indicates the general formula.
  - In each time unit, *one move plus local computations take place.*
- Each dot represents one time unit.
- The y_i values are computed as they flow up through the array of PEs.
- Each x_i value is computed at P_1 and its value is used in the recursive computation of the y_j values at each P_k as x_i flow downward through the array of processors.
- Elements of A reach the PEs where they are needed at the appropriate time.

h. General Algorithm - Input to Array:

- The sequence $y_1, y_2, \ldots, y_n$ is initialized successively to 0 in P_n, separated by one time delay.
- The sequence of i^{th} diagonal elements of A (starting with its main diagonal and continuing with the diagonals below the main diagonal), namely

$$a_{i1}, a_{i+1,2}, \ldots, a_{n,n-i+1}$$

  are fed into P_i, one element at a time, separated by one time delay. The first input starts after a delay of $n + i - 2$ time units.
- The elements $b_1, b_2, \ldots, b_n$ are fed into P_1, separated by one time unit delay. This input starts after a delay of $n - 1$ time units.
- The elements of $x_1, x_2, \ldots, x_n$ are successively defined in P_1,

separated by one unit time delay. This input starts after a delay of $n - 1$ time units.

- When x_i reaches P_n, it exits the array as output.

i. **General Algorithm - Computation in Array:**

- The values x_i, a_{ii}, and b_i simultanenously arrive at P_1 and the (final) value of x_i is computed as follows:

$$x_i \leftarrow (b_i - y_i)/a_{ii}$$

- At P_1, $y_0 = 0$ and y_i (for $i > 1$) is equal to

$$a_{i1}x_1 + a_{i2}x_2 + \ldots + a_{i,i-1}x_{i-1}$$

This ensures that

$$x_i = (b_i - \sum_{j=1}^{i-1} a_{ij}x_j)/a_{ii},$$

which is the desired value.

- In the processor P_k for

$2 \le k \le n$, the elements a_{ij}, x_j, and y_i arrive at the same time and P_k performs the following computation:

$$y_i \leftarrow y_i + a_{ij}x_j$$

At this point, $k = i - j + 1$.

j. **First few steps of algorithm for n = 4** (See Figure 7.7 in Akl's book on pg 287)
   - In each step, some local computation and a move may occur.
   - At time $u = 0$, the initial input begins. Note that y_1 is set to 0 in P_4.
   - At time $u = 3$ (column a), the values y_1, a_{11}, b_1 reach P_1 and are used to define x_1 as

     $$x_1 \leftarrow (b_1 - y_1)/a_{11} = b_1/a_{11}$$

   - At time $u = 4$ (column b), value x_1 reaches P_2 and is used to update y_2

$$y_2 \leftarrow y_2 + a_{21}x_1 = a_{21}x_1$$

- At time $u = 5$ (column c), values $y_2, a_{22},\ b_2$ reach P_1 and are used to define x_2 as

$$x_2 \leftarrow (b_2 - y_2)/a_{22} = (b_1 - a_{21}x_1)/a_{22}$$

  Additionally, value x_1 reaches P_3 and is used to update y_3 as follows:

$$y_3 \leftarrow y_3 + a_{31}x_1 = a_{31}x_1$$

- Value x_1 is output at $u = 5$ and x_2 is output at $u = 7$.
- Note that in Figure 7.7, only half of the processors are active at any time.

k. See Figure 7.7 on page 287 of Akl's textbook

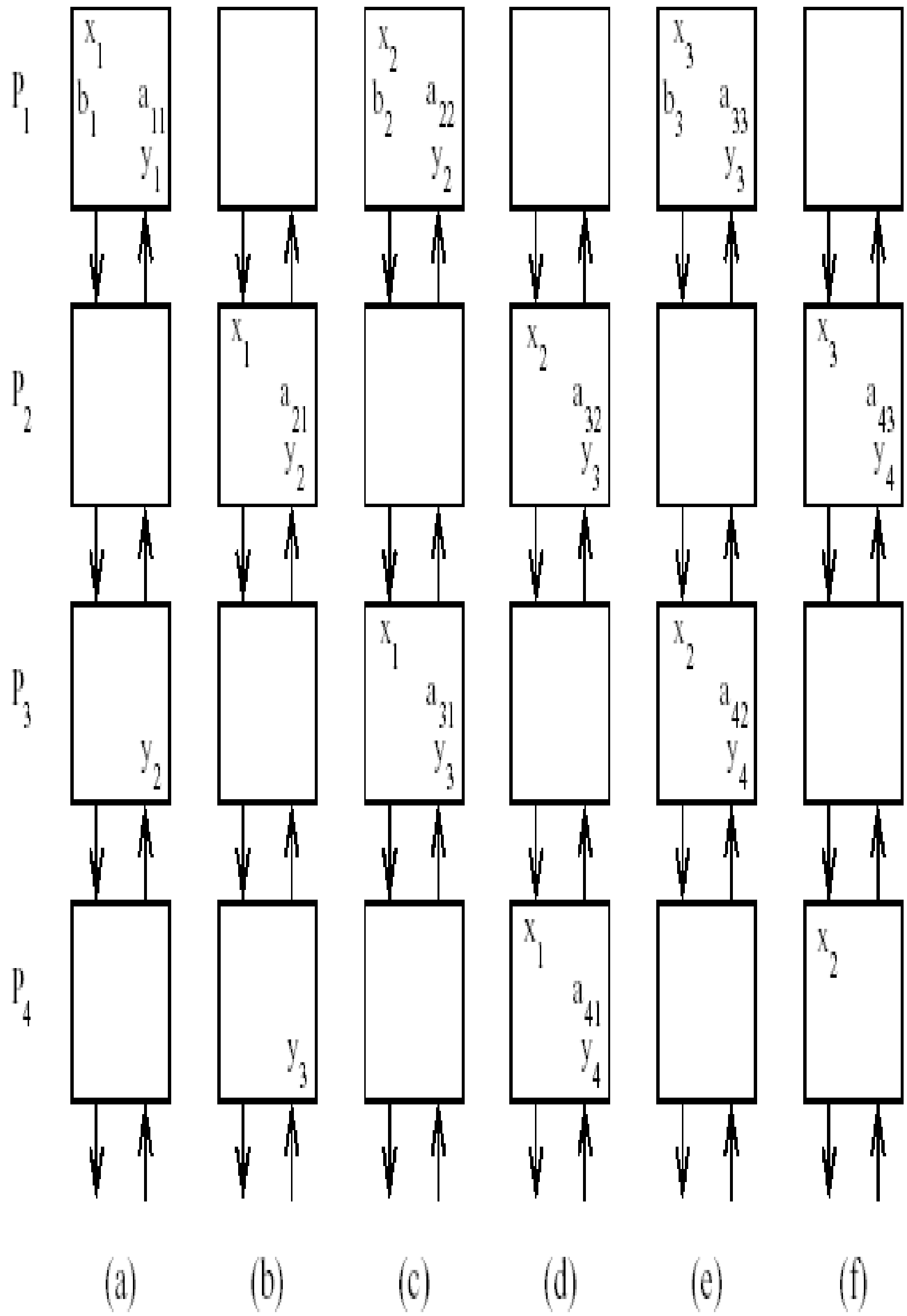

Figure 7.7: Solving a triangular system of equations on a linear array: (a) $u = 3$; (b) $u = 4$; (c) $u = 5$; (d) $u = 6$; (e) $u = 7$; (f) $u = 8$.

l. Algorithm Analysis:

- y_1 reaches P_1 in $n - 1$ time units.
- n time units later, x_1 is output by P_n.
- Each remaining element of vector x is output at intervals of 2.
- $t(n) = (n - 1) + n + 2(n - 1)$ $= 4n - 3$.
- $c(n) = (4n - 3)(n) = 4n^2 - 3n$ or $\theta(n^2)$ which is optimal.

m. Some Possible Time Improvement:

- x_i can be output by P_1, while a copy travels down the array, saving $n - 1$ steps at the conclusion of the algorithm.
  - Recomputing above timing yields $t^*(n) = t(n) - (n - 1) = 3n - 2$
  - Additionally, there is no need to initially wait $n - 1$ steps for y_1 to reach P_1,

reducing the time to

$$t^{**}(n) = 2n - 1$$

- Another possible variation: The b values can be fed to P_n instead of P_1.
  - Then, y_i is initialized to b_i and the computation in P_k for $k > 1$ becomes

    $$y_i \leftarrow y_i - a_{ij}x_j.$$

  - The computation in P_1 becomes

    $$x_i \leftarrow y_i/a_{ii}$$

- The utilization of PEs can be significantly improved by using an array of $n/2$ PEs and have each simulate two PEs in the algorithm

Possible Lecture Topics

1. **Convolutions**
   a. Setting: Let
      - $W = \{w_1, w_2, \ldots, w_k\}$ be a sequence of weights.
      - $X = \{x_1, x_2, \ldots, x_n\}$ be an input sequence.
   b. The required output is the sequence

$$Y = \{y_1, y_2, \ldots, y_{n+1-k}\}$$

where

$$\begin{aligned}
y_1 &= w_1 x_1 + w_2 x_2 + \ldots + w_k x_k \\
y_2 &= w_1 x_2 + w_2 x_3 + \ldots + w_k x_{k+1} \\
\ldots &= \ldots \\
y_i &= w_1 x_i + w_2 x_{i+1} + \ldots + w_k x_{i+k-1} \\
\ldots &= \ldots \\
y_{n+1-k} &= w_1 x_{n+1-k} + \ldots + w_k x_n
\end{aligned}$$

   c. In particular, $Y = \{y_1, y_2, \ldots, y_{n+1-k}\}$ where

$$y_i = \sum_{j=1}^{k} w_j x_{i+j-1}$$

d. **Example 7.4 and Figure 7.8:** Suppose we have 3 weights $\{w_1, w_2, w_3\}$ and 8 inputs $\{x_1, x_2, \ldots, x_8\}$. Then we may slide one sequence past the other to produce the output $\{y_1, y_2, \ldots, y_6\}$ as follows:

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
y_1	w_1	w_2	w_3					
y_2		w_1	w_2	w_3				
y_3			w_1	w_2	w_3			
y_4				w_1	w_2	w_3		
y_5					w_1	w_2	w_3	
y_6						w_1	w_2	w_3

e. Sequentially, the sequence Y can be computed in

$$(n + 1 - k) \times k = \theta(nk) \text{ time}$$

f. Four Algorithm Approaches in Text:

- There are 3 data arrays:
  - The input array
  - The weight array
  - The output array being computed
- Items in two of these data types march across the array of PEs.
- Items in the remaining data type are initially assigned to a specific PE.
- The data items that move can either move in the same or opposite directions

g. Algorithm 1: Input and Weights travel in opposite directions.

$$.x_2.x_1 \rightarrow \begin{bmatrix} P_3 \\ y_3 \end{bmatrix} \rightleftarrows \begin{bmatrix} P_2 \\ y_2 \end{bmatrix} \rightleftarrows \begin{bmatrix} P_1 \\ y_1 \end{bmatrix} \leftarrow \ldots . w_1.$$

- There is one PE for each weight.
- The k weights are fed to P_1,

separated by one time delay.

- There are $k - 1$ delays initially before w_1 is fed to P_1 so that w_1 and x_1 reach P_1 at the same time.
- After last weight w_k is fed to P_1, the weights recycle, starting with w_1.

- The inputs $x_1, x_2, \ldots, x_n$, separated by a time delay, are fed to P_k.
- Each processor P_i holds the current value of y_i, which is initially zero.
- Note that each P_i receives an x-value and a w-value every other time unit.
- Each time an x-value meets a w-value in P_i, their product is computed and added to y_i.
- When the computation of y_i is finished, it is output on the x-line in the gap between

x-values.

- The value y_i is computed as soon as w_k is included in the computation.
  - w_k is identified by a special tag
- As soon as a PE completes the computation of y_i, the computation of y_{i+k} starts, provided $i + k \leq n + 1 - k$.

h. Example for Algorithm 1:

Example 7.5 and Expanded Fig 7.11

i. Analysis for Algorithm 1:

- Let $q = (n + 1 - k) \bmod k$
- Let P_i be the last processor to output.
- If $q = 0$, then $n + 1 - k$ is a multiple of k and $i = k$ so P_k outputs last.
- If $q \neq 0$, then $i = q$ and P_q outputs last.
  - Comment: In Example 7.5

and Fig. 7.11,
$n + 1 - k = 5 + 1 - 3 = 3$, so
$q = 3 \bmod 3 = 0$ and y_3 is last y computed and is computed at P_3.

- x_n will enter P_k at time $2n - 1$ due to delays.
- The distance from P_k to P_i is $k - i$, so x_n enters P_i at time $(2n - 1) + (k - i)$.
- Output from P_i takes $(i - 1)$ time units.
- Total time required is $(2n - 2) + k$.
- Note that on average, only one-half of the k processors are performing computation during a time unit.

j. **Algorithm 2:** Inputs and weights travel in the same direction.

$$\begin{array}{l} \ldots w_1 w_3 w_2 w_1 \\ \ldots\ldots x_4 x_3 x_2 x_1 \end{array} \begin{array}{c} \rightarrow \\ \rightarrow \end{array} \left| \begin{array}{c} y_1 \\ P_1 \end{array} \right| \rightrightarrows \left| \begin{array}{c} y_2 \\ P_2 \end{array} \right| \rightrightarrows \left| \begin{array}{c} y_3 \\ P_3 \end{array} \right.$$

- Weights and inputs at processor P_1 travel in the same direction.
- The x-values travel twice as fast as the w-values, with each w-value remaining inside each processor an extra time period.
- When all the w-values have been fed to P_1, the w-values are recycled.
- Each time a x-value meets a w-value in a processor, their product is computed and added to the y-value computed by the processor.
- When a processor finishes the computation of y_j, it
  - places the value of y_j in the gap between w-values so

that it will be output at P_n.

    - begins the computation of y_{j+k} at the next step if $j + k \leq n + k - 1$.

- A processor computes each step until its computation is finished.
- The convolution of k weights and n inputs requires $n + k - 1$ time units.

k. Algorithm 3: Input and Outputs travel in opposite directions:

$$.x_2.x_1 \rightarrow \begin{bmatrix} P_3 \\ w_3 \end{bmatrix} \leftrightarrows \begin{bmatrix} P_2 \\ w_2 \end{bmatrix} \leftrightarrows \begin{bmatrix} P_1 \\ w_1 \end{bmatrix} \leftarrow y_1.y_2$$

- The value w_i is stored in processor P_i.
- The x-values are fed to P_k and march across the array from left to right.
- The y-values are fed to P_1 and are initialized to 0, then march across the array from right to left.

- Consecutive x-values and consecutive y-values are separated by 2 time units.
- A processor performs a computation only when an x-value meets a y-value.
- Convolution of k weights and n inputs requires $2n - 1$ time units.

l. **Algorithm 4:** Inputs and outputs travel in the same direction:

$$\begin{array}{l} \ldots y_1 y_3 y_2 y_1 \\ \ldots\ldots x_4 x_3 x_2 x_1 \end{array} \begin{array}{c} \rightarrow \\ \rightarrow \end{array} \left| \begin{array}{c} w_1 \\ P_1 \end{array} \right| \rightrightarrows \left| \begin{array}{c} w_2 \\ P_2 \end{array} \right| \rightrightarrows \left| \begin{array}{c} w_3 \\ P_3 \end{array} \right.$$

- The value w_i is stored in processor P_i.
- y-values march across the array from left to right.
- x-values march across the array from left to right at one-half the speed of the y-values.

    - Each x-value is slowed down by being stored in a processor register every other time unit.
- Each time a x-value meets a y-value, the product of the x-value and the w-value is computed and added to the y-value.
- Convolution of k-weights with n-inputs requires $n + k - 1$ time.