Hypercubes

(Chapter Nine)

Mesh Shortcomings:

- Due to its simplicity and regular structure, the mesh is attractive, both theoretically and practically.
- A problem with the mesh is that movement of data is inherently slow, due to its large diameter of $O(\sqrt{n})$.
- However, optimal algorithms for many problems that does not involve long data movements (e.g., image processing) can be designed for a mesh.
- Approaches to increasing communication speed (including data movement):
  - Going to a higher dimensional mesh will provide some speedup.
    - E.g., the diameter of the 3D mesh is $O(\sqrt[3]{n})$, which is smaller.

- Modify mesh by replacing or augmenting some of its edges.
  - Mesh of Trees (see Chapters 2 & 8 of Akl's textbook)
  - Pyramid (see Chapters 2 & 8 of Akl's textbook)
- Modify the mesh by adding buses (see Ch. 10 of Akl's textbook).
- Use a different network with a smaller diameter.
  - Hypercube
  - Perfect Shuffle
  - Cube-Connected cycles
  - de Bruijn
  - Star

Hypercube Features

- The diameter of the hypercube is $O(\lg n)$.
- Hypercubes of dimension 0, 1, 2, 3, and 4 are shown in Figure 2.16 of Akl's textbook.
- A hypercube of dimension $g \geq 0$ has $N = 2^g$ processors $P_0, P_1, \ldots, P_{N-1}$.

- Each index i for a processor P_i can be expressed in binary as follows:

$$i = i_{g-1} i_{g-2} \ldots i_0$$

- For $b = 0, 1, \ldots, g-1$, let index $i^{(b)}$ have the same binary representation as i except that its bit b is complemented.
- Then,

$$i^{(b)} = i_{g-1} i_{g-2} \ldots i_{b+1} i_b' i_{b-1} \ldots i_0$$

  where i_b' is the binary complement of i_b.
- A g-dimensional hypercube connects processor P_i to processors $P_{i^{(0)}}, P_{i^{(1)}}, \ldots, P_{i^{(g-1)}}$ using a 2-way link.i

Prefix Operations on the Hypercube

- Sometimes called ”Recursive Doubling”
- Let $P_0, P_1, \ldots, P_{n-1}$ be a hypercube with a number x_i in P_i.
- For $0 \leq i \leq n-1$, assume each P_i has two registers A_i and B_i.
- Initially both A_i and B_i contain x_i.
- Let $*$ be an associative binary operation

(e.g., +, ×, min, max)

- When the algorithm terminates, A_i contains $x_0 * x_1 * \ldots * x_i$.
- Hypercube Prefix Operations Algorithm (See Example 2.5)
  - For $j = 0$ to $(\log n) - 1$ do
    - For all i with $0 \leq i \leq n - 1$ and $i < i^{(j)}$ do in parallel
      1. $A_{i^{(j)}} \leftarrow A_{i^{(j)}} * B_i$
      2. $B_{i^{(j)}} \leftarrow B_{i^{(j)}} * B_i$
      3. $B_i \leftarrow B_{i^{(j)}}$
    - end for
    - end for
- Figure 2.21 illustrates the prefix sum algorithm for $n = 8$.
  - A_i and B_i are shown as the top and bottom registers of P_i.
  - X_{ij} denotes $x_i + x_{i+1} + \ldots + x_j$.
  - When the algorithm terminates, $A_i = X_{0i}$while $B_i = X_{0,n-1}$.

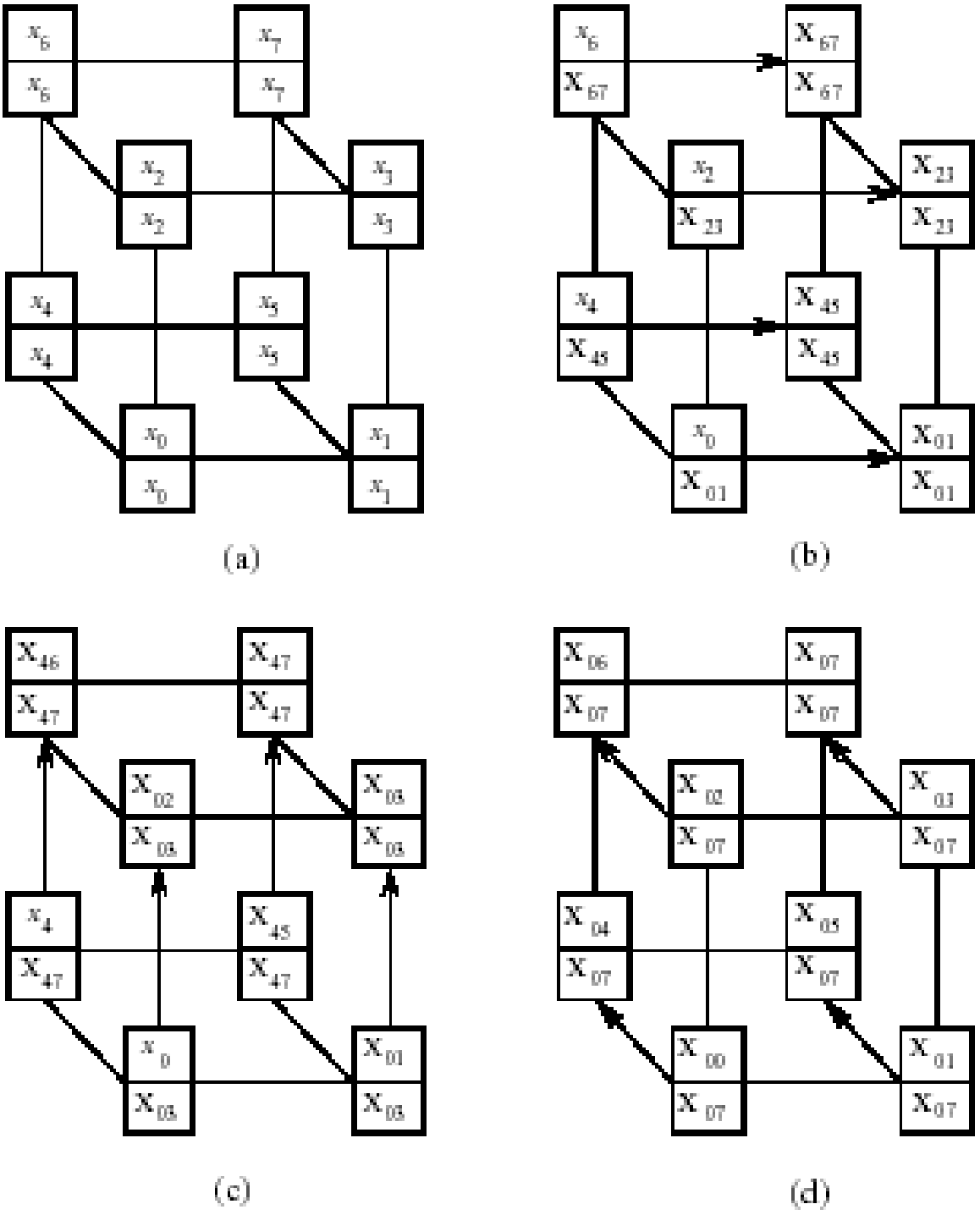


Figure 2.21: Computing the prefix sums on a hypercube: (a) Initially; (b) $j = 0$; (c) $j = 1$; (d) $j = 2$.

- The running time of the prefix operations algorithm is $t(n) = O(\log n)$
- Its cost is $c(n) = O(n \log n)$.
- While its cost is not optimal, an optimal algorithm can be designed, as with the PRAM prefix sum, but this may require the use of ”partial hypercubes”.

Sequential Matrix Products

- Let A and B be $n \times n$ matrices and $C = A \times B$.
- A lower bound on the number of operations required to compute C is $\Omega(n^2)$, as any algorithm must produce the n^2 values in C.
  - This is also the best (i.e., largest) known *lower* bound.
- Currently, the fastest known RAM algorithm for calculating C requires $O(n^{2.38})$ time.
- Since C has n^2 entries and each entry is a

sum of n products, a straight-forward RAM algorithm requires $O(n^3)$ time.

Parallel Matrix Multiplication

1. Why Choose Matrix Multiplication?
   - Matrix multiplication is used as an example of a typical algorithm on the hypercube model.
   - Nothing is special about matrix multiplication that favors its implementation on the hypercube.
   - Its inclusion here allows the usefulness of parallel matrix multip lication in other algorithms to be demonstrated later in this chapter.
2. Let A and B be $n \times n$ matrices and $C = A \times B$.
3. The algorithm given assumes that n is a power of 2 and uses a hypercube with $N = n^3$ PEs.
   - **Aside:** In cases where n is not a power of 2, we either
     - use n' in place of n where n' is

the smallest power of 2 with $n' > n$.

- use the theory of ”partial hypercubes” to minimize the number of processors needed.

- N and n are powers of 2 and may be expressed as $n = 2^g$ and $N = 2^{3g}$.
- PEs will be visualized in 3D-space in a $n \times n \times n$ array with processor P_r in position (i,j,k), where

$$r = in^2 + jn + k$$

and $0 \leq i,j,k < n$.

- The processor P_r at position (i,j,k) is denoted as $P(i,j,k)$.
- If $P_r = P(i,j,k)$ and r has the binary representation

$$r = r_{3g-1}r_{3g-2} \ldots r_1 r_0$$

then the indices i,j,k and r are related as follows:

$$i = r_{3g-1} r_{3g-2} \ldots r_{2g}$$

$$j = r_{2g-1} r_{2g-2} \ldots r_{g}$$

$$k = r_{g-1} r_{g-2} \ldots r_{0}$$

- All PEs with the same fixed value in one of the 3 coordinates (i.e., i, j, k) form a hypercube of n^2 PEs.
- All PEs with fixed values in 2 fixed coordinates (i.e., i, j, k) form a hypercube of n PEs.
- Assume each P_r has three registers A_r, B_r, and C_r, which are denoted by

  $A(i,j,k)$, $B(i,j,k)$, and $C(i,j,k)$.

- Initially, the processors in the base plane contain the matrices A and B. In particular,
  - register $A(0,j,k)$ contains the element a_{jk} of matrix A
  - register $B(0,j,k)$ contains the element b_{jk} of matrix B.
  - All other registers are

initialized to 0.

- At the end of the computation, matrix C is stored in the base plane with c_{jk} in $C(0,j,k)$.

4. Algorithm **Hypercube Matrix Multiplication** has three major steps.

a. Distribute the elements of A and B over the n^3 PEs so that $A(i,j,k)$ contains a_{ji} and $B(i,j,k)$ contains b_{ik} for all i, j, and k, as follows:

- Copy $A(0,j,k)$ plane to $A(i,j,k)$ so that $A(i,j,k) = a_{jk}$ for all i using recursive doubling in $O(\lg n)$ time.
  - The nr. of planes copied doubles at each step.
- Likewise, copy plane $B(0,j,k)$ to $B(i,j,k)$ for all i so that $B(i,j,k) = b_{jk}$.
- On each plane i, perform the next step so that $A(i,j,k) = a_{ji}$ for all j and k with $0 \le j,k < n$.
  - On i^{th}plane, copy the

i^{th} column into each column.

- Takes $\lg n$ steps using recursive doubling.
- These actions occur on hypercubes of n^2 PEs
- Look at subsequent 3D diagram of PEs to follow this action geometrically.

- On each plane i, perform the next step so that $B(i,j,k) = b_{ik}$ for all j and k with $0 \leq j,k < n$.
  - On i^{th}plane, copy the i^{th} row into each row.
  - Takes $\lg n$ steps using recursive doubling.
  - The actions occur on hypercubes of n^2 PEs
  - Look at subsequent 3D diagram of PEs to follow this action geometrically.

b. Each PE on plane i computes the product

$$A_r \times B_r = a_{ji} \times b_{ik}$$

and places result in C_r in one time step.

c. Compute the sum

$$C(0,j,k) \leftarrow \sum_{i=0}^{n-1} C(i,j,k)$$

for $0 \leq j,k \leq n-1$ as follows:

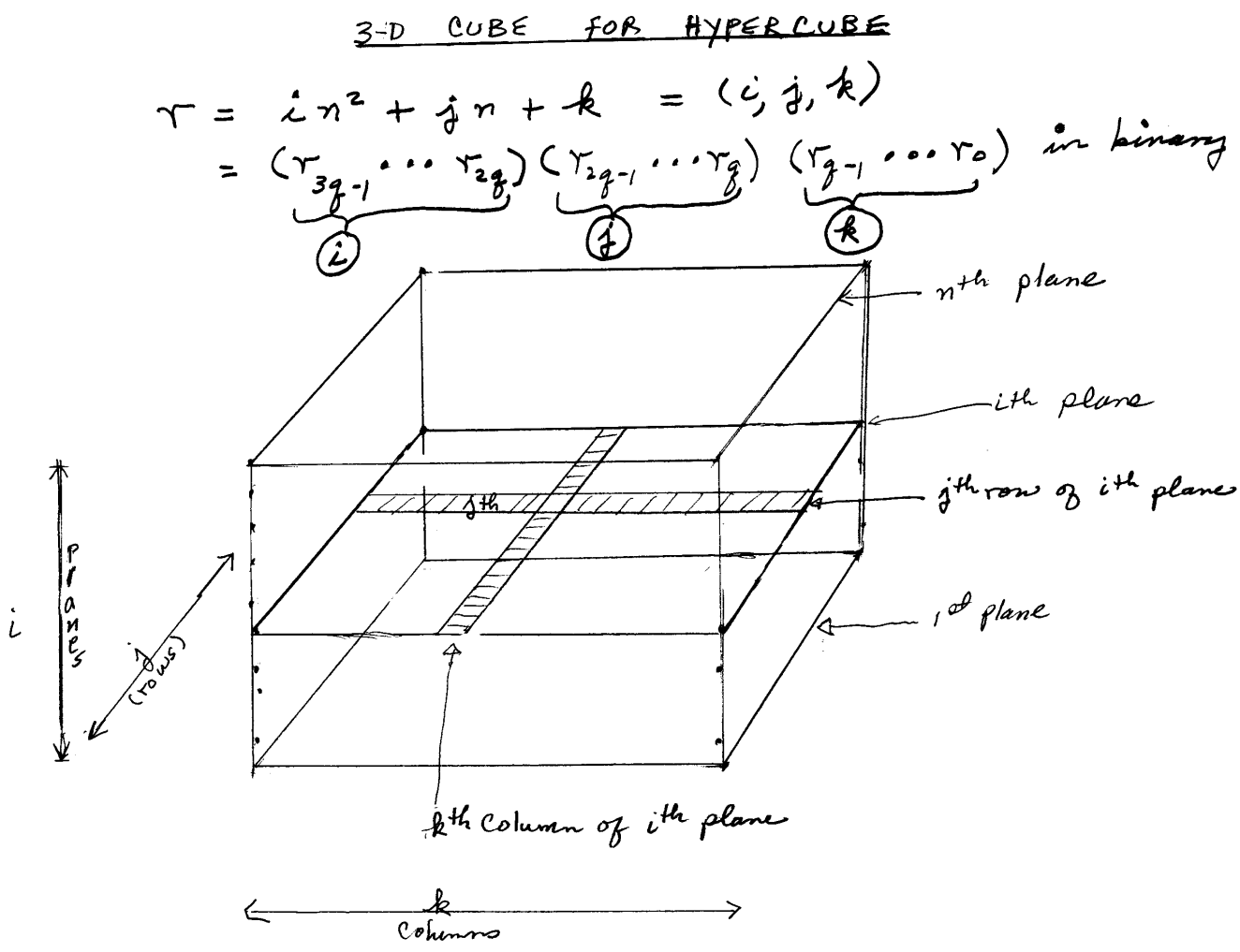
- The values $C(i,j,k)$ to be summed to get c_{jk} have been stacked vertically above position $(0,j,k)$.
- All prefix sums from top to bottom are computed simultaneously and placed in position $C(0,j,k)$
  - This procedure is somewhat the reverse of the recursive doubling procedure for planes
- These sums are calculated on hypercubes of n PEs.

- This step takes $\lg n$ steps.

5. **Analysis:** The algorithm takes $O(\lg n)$ steps using n^3 PEs.

   The cost of $O(n^3 \lg n)$ is non-optimal, but the algorithm is very fast.

6. Diagram for 3-D visualization of the hypercube



Matrix Transposition

- Definitions:
  - Let $A = (a_{ij})$ be an $n \times n$ matrix.
  - The *transpose* of A is

$$A^T = (a_{ij}^T)$$

where $a_{ij}^T = a_{ji}$ for $1 \leq i,j \leq n$.

- Hypercube Setup for Computing Transpose
  - We assume n is a power of 2, say $n = 2^q$
  - The hypercube used has $N = n^2 = 2^{2q}$ processors
  - **Aside:** In cases where n is not a power of 2, we either

1.
   - use n' in place of n where n' is the smallest power of 2 with $n' > n$.
   - use the theory of 'partial hypercubes' to minimize the number of processors needed.

- This hypercube is identified with an $n \times n$ mesh and P_r is denoted $P(i,j)$ where

$$r = in + j \text{ and } 0 \leq i,j \leq n-1$$

  - If the binary representation of r is

$$r = r_{2q-1}r_{2q-2}\ldots r_q r_{q-1}\ldots r_0$$

then binary representation of i and j is the first half and second half of r's representation, namely

$$i = r_{2q-1}r_{2q-2}\ldots r_q$$

$$j = r_{q-1}r_{q-2}\ldots r_0$$

- When the algorithm terminates, a_{ij} is stored in $P_s = P(j,i)$ with $s = jn + i$.
- Note that a_{ij} can be routed from $P_r = P(i,j)$ to $P(j,i)$ with at most $2q$ steps.
  - Check one bit of r each step and complement it, if needed.
  - The algorithm will follow a different, recursive movement

- Algorithm Preliminaries:
  - Each processor P_r has registers A_r and B_r.
  - Register A_r initially holds a_{ij} where $r = in + j$ and $0 \leq i,j < n$.
  - When the algorithm terminates, A_r

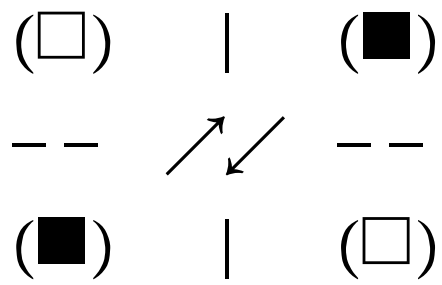
holds $a_{ij}^{T} = a_{ji}$.

- Recall binary indexes $r^{(b)}$ and r differ only in position b.
- P_r uses an additional register B_r for routing data.
- The algorithm is easily stated but difficult to interprete. A simple recursive explanation is given below.

- Hypercube Matrix Transpose (A)

  For $m = 2q - 1$ down to q do
  For $u = 0$ to $N - 1$ do in parallel
  (1) If $u_m \neq u_{m-q}$
  then $B_{u^{(m)}} \leftarrow A_u$
  (2) If $u_m = u_{m-q}$
  then $A_{u^{(m-q)}} \leftarrow B_u$

- Recursive Explanation of Action of Algorithm
  - Divide an $n \times n$ matrix into four $\frac{n}{2} \times \frac{n}{2}$ submatrices

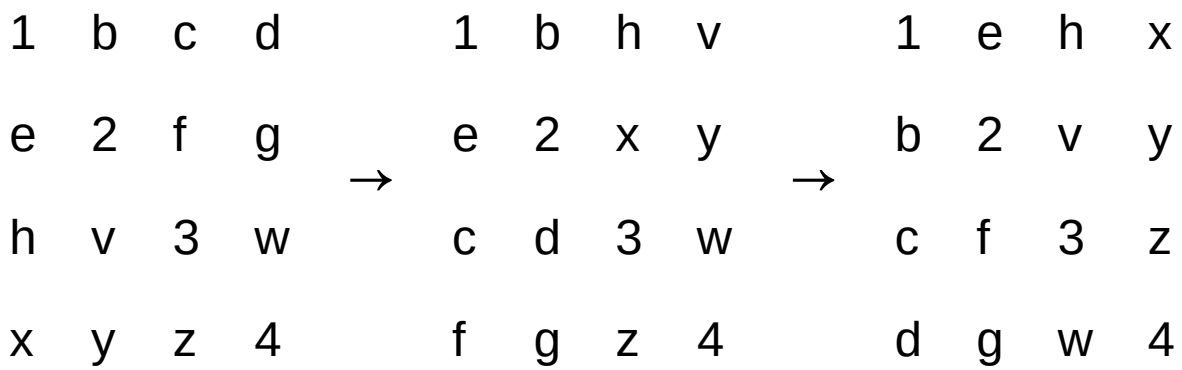


- The first iteration of algorithm (i.e, $m = 2q - 1$) swaps the corresponding elements in the upper right and lower left submatrices, leaving other submatrices untouched.
  - Step (1) moves each element in the A register of darken matrices to the corresponding B register of the undarken matrix directly above or below it.
  - Step (2) moves each element in the B register of the undarken matrix to the corresponding darken matrix to its right or left
- In each successive iteration,

proceed recursively with the matrices created in the previous iteration.

- The number of submatrices increase by a factor of 4.
- All submatrices are processed in parallel.
- Each submatrix is processed using the same procedure used in first iteration. (e.g., divide each into four submatrices, etc.)

- An example on pg 372-3 of Akl's textbook gives full details. The initial matrix and the one produced at the end of each of the two interations are as follows:



- Analysis of Matrix Transposition
  - Algorithm consists of q constant time iterations, so $t(n) = O(\lg n)$.
  - Since $p(n) = n^2$, the cost $c(n) = O(n^2 \lg n)$.
  - The cost is not optimal.
    - To see this, note that there are $n^2 - n = n(n-1)$ elements off the main diagonal.
    - Swapping each pair of elements a_{ij} and a_{ji} off the main diagonal sequentially requires $n(n-1)/2$ or $O(n^2)$ swaps.

Computing the Connectivity Matrix

- Recall the *adjacency matrix* of a graph G with n vertices is an $n \times n$ matrix $A = (a_{ij})$ defined by

$$a_i = \begin{cases} 1 & \text{if } v_i \text{ is connected to } v_j \\ 0 & \text{otherwise} \end{cases}$$

- The *connectivity matrix* of a (directed or undirected) graph G with n vertices is an $n \times n$ matrix $C = (c_{ij})$ defined by

$$c_{ij} = \begin{cases} 1 & \text{if } \exists \text{ a path from } v_i \text{ to } v_j \\ 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

- **Problem:** Given the adjacency matrix A, determine how to compute the connectivity matrix C.
- An algorithm to compute the adjacency matrix will be developed next.
- **Boolean Matrix Multiplication:**
  - Defined for binary matrices
  - Product is the same as for ordinary

matrices, except that the binary operations "×" and "+" are replaced with "**and**" (denoted $\wedge$) and "**or**" (denoted $\vee$), respectively.

- Binary Operation Facts:

$$0 \wedge 0 = 0 \wedge 1 = 1 \wedge 0 = 0$$

$$1 \wedge 1 = 1$$

$$0 \vee 0 = 0$$

$$0 \vee 1 = 1 \vee 0 = 1 \vee 1 = 1$$

- Define $n \times n$ boolean matrix $B = (b_{ij})$ as follows: B is identical to the adjacency matrix A except that it has 1's on its main diagonal.
  - Note that an element b_{ij} equals 1 if and only if there is a path from v_i to v_j of length ≤ 1.
- The matrix B is said to represent all paths of length at most 1.
- Matrix B^2 represents all paths of length at most 2. See Figure 9.5 from Text below:

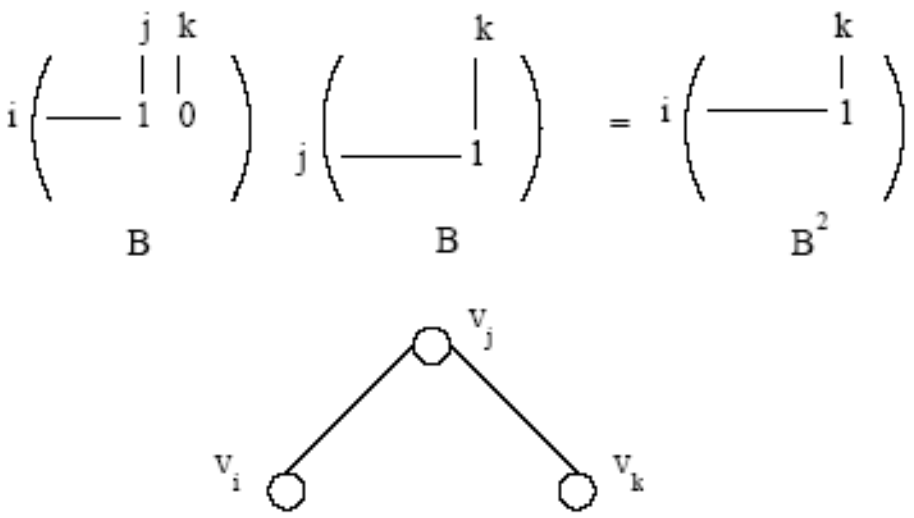


Figure 9.5: A path from v_i to v_k goes through v_j.

- Matrix B^3 represents all paths of length (by same argument as in Fig. 9.5 for $B^2 \times B$.)
- In general, B^k represents all paths of length at most k.
- Since graph has only n vertices, each path can be assumed to have length at most $n - 1$; hence,

$$B^k = C \text{ for all } k \geq n - 1$$

- Let $k = \lceil \lg(n - 1) \rceil$. Then k is the smallest integer with $2^k \geq n - 1$.
- **Hypercube Connectivity Algorithm**
  - The connectivity matrix C is computed using $k = \lceil \lg(n - 1) \rceil$

binary matrix products, as follows:

$$B^2 \leftarrow B \times B$$
$$B^{2^2} \leftarrow B^2 \times B^2$$
$$B^{2^3} \leftarrow B^{2^2} \times B^{2^2}$$
$$\ldots\ldots\ldots\ldots\ldots\ldots$$
$$B^{2^k} \leftarrow B^{2^{k-1}} \times B^{2^{k-1}}$$
$$C \leftarrow B^{2^k}$$

- Please study the pseudocode description of this algorithm on pg 377 of Akl's textbook.

- The Hypercube Matrix Multiplication Algorithm used here requires that the matrix being squared be stored in both the $A(0,j,k)$ and $B(0,j,k)$ registers prior to each iteration.
- The product matrix obtained in each iteration has its values stored in the $C(0.,j,k)$ register.
- Analysis of Algorithm Hypercube Connectivity:
  - A modified version of Hypercube Matrix Multiplication is used

$\lceil \lg(n-1) \rceil$ times.

- Since the Hypercube Matrix Multiplication runs in $O(\lg n)$ time, the running time of the Hypercube Connectivity Algorithm is $O(\lg^2 n)$
- Since $p(n) = n^3$, its cost is $O(n^3 \lg^2 n)$

Connected Components

- We assume that G is a undirected graph.
- Consider row i of the connectivity matrix C.
  - It identifies all of the vertices v_j that are connected to v_i by a path.
  - Then all the vertices connected to v_i (i.e., as identified by the 1's in row i) is a connected component in the graph.
  - Let l be the smallest index with $c_{il} = 1$. Then "l" is the name given to the connected component in the graph containing v_i.
- We define an $n \times n$ matrix D as follows:

$$d_{ij} = \left\{ \begin{array}{ll} v_j & \text{if } c_{ij} = 1 \\ 0 & \text{otherwise} \end{array} \right]$$

- Connected Component Implementation Details:
  - Assume the same registers as for the Hypercube Connectivity Algorithm.
  - The matrix D is computed in the registers $C(0,j,k)$ in processors $P(0,j,k)$.
  - Initially, $A(0,j,k)$ contains the adjacency matrix entry a_{jk}.
  - At the end of Step 2 of the computation,
    - $D(j,k)$ is stored in $C(0,j,k)$.
    - $D(j,0) = C(0,j,0)$ contains the component number for the vertex v_j.
    - For each $k > 0$, $D(j,k)$ contains the vertices in the component for vertex v_j.
      - Clearly, vertex v_0 is in the

component for vertex v_j if $D(j,0) = 0$.

- The $n \times n$ matrix E is defined to be identical to D except for the first column. The value $E(j,0)$ in the first column of row j contains the label name for the connected component in row j.
  - Clearly, v_0 is in the first column of $D(j,0)$ exactly when $E(j,0) = 0,$ so E contains all of the information in D, but is more useful.
  - At the end of Step 3 of the matrix entry $E(j,k)$ is stored in register $C(0,j,k)$.

- **Hypercube Connected Components** $(\mathbf{A},\mathbf{C})$

  Step 1: Hypercube Connectivity (A,C)

  Step 2: For $j = 0$ to $n-1$ do in parallel

  For $k = 0$ to $n-1$ do in parallel

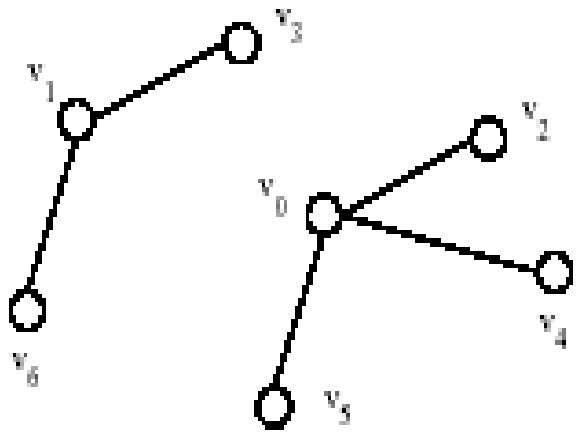
if $C(0,j,k) = 1$
then $C(0,j,k) \leftarrow v_k$

Step 3: For $j = 0$ to $n - 1$ do in parallel

(i) The n processors in row j find the smallest l with $C(0,j,l) \neq 0$

(ii) $C(0,j,0) \leftarrow l$

- Figure 9.6 follows:



(a)

	0	1	2	3	4	5	6
0	0	0	1	0	1	1	0
1	0	0	0	1	0	0	1
2	1	0	0	0	0	0	0
3	0	1	0	0	0	0	0
4	1	0	0	0	0	0	0
5	1	0	0	0	0	0	0
6	0	1	0	0	0	0	0

(b)

	0	1	2	3	4	5	6
0	1	0	1	0	1	1	0
1	0	1	0	1	0	0	1
2	1	0	1	0	1	1	0
3	0	1	0	1	0	0	1
4	1	0	1	0	1	1	0
5	1	0	1	0	1	1	0
6	0	1	0	1	0	0	1

(c)

	0	1	2	3	4	5	6
0	v_0	0	v_2	0	v_4	v_5	0
1	0	v_1	0	v_3	0	0	v_6
2	v_0	0	v_2	0	v_4	v_5	0
3	0	v_1	0	v_3	0	0	v_6
4	v_0	0	v_2	0	v_4	v_5	0
5	v_0	0	v_2	0	v_4	v_5	0
6	0	v_1	0	v_3	0	0	v_6

(d)

Figure 3.6: Computing connected components on a hypercube: (a) Graph with two connected components: (b) Adjacency matrix: (c) Connectivity matrix: (d) Matrix of connected components.

- Note that matrix E can be obtained from matrix D in Figure 9.6 by replacing the first column of D with the vector $(0, 1, 0, 1, 0, 0, 1)$ of component labels.
- **Time and Cost Analysis**
  - Step 1 takes $O(\lg^2 n)$ time
  - Step 2 takes constant time
  - The first part of Step 3 involves finding a minimum value in a hypercube with n elements.
    - Uses the hypercube prefix minimium operation along each row j in processors $P(0, j, k)$.
      - Same as Step 3 of Hypercube Matrix Multiply, except use min instead of +.
      - Also, see Hypercube Prefix Sum in section 2.3
    - Takes $O(\lg n)$ time
  - The second part of Step 3 takes constant time.
  - The running time is $t(n) = O(\lg n)$
  - Since $p(n) = n^3$, the cost is

$$c(n) = O(n^3 \lg^2 n)$$

COMMENTS ABOUT PRESENTATION:

- Some of the details in this presentation have been suppressed so that while the basic concepts are captured, the time spent trying to figure out details such as the precise representation of indices, etc.is minimized.
- The algorithms are captured at a sufficiently low level to demonstrate real algorithm and not just a high-level overview of it.
- More details are included in textbook coverage.