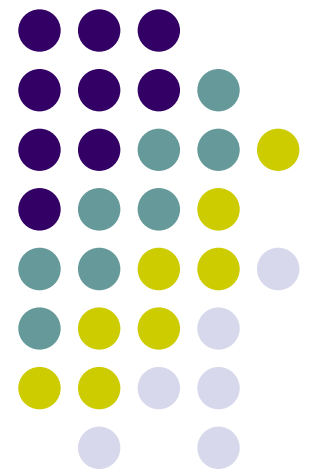


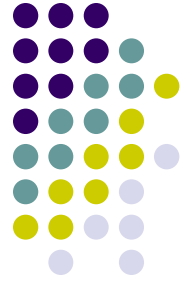
CSCI 1950

Intro to Computational Complexity

Parallel Computation II

John E Savage



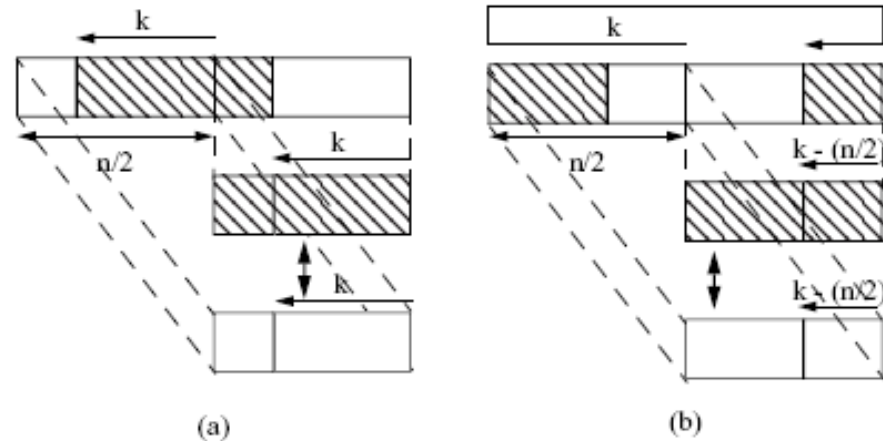


Overview

- Cyclic shift on hypercubes
- Shuffle operations on linear arrays
- Shuffle operations on two-dimensional arrays
- Routing in networks



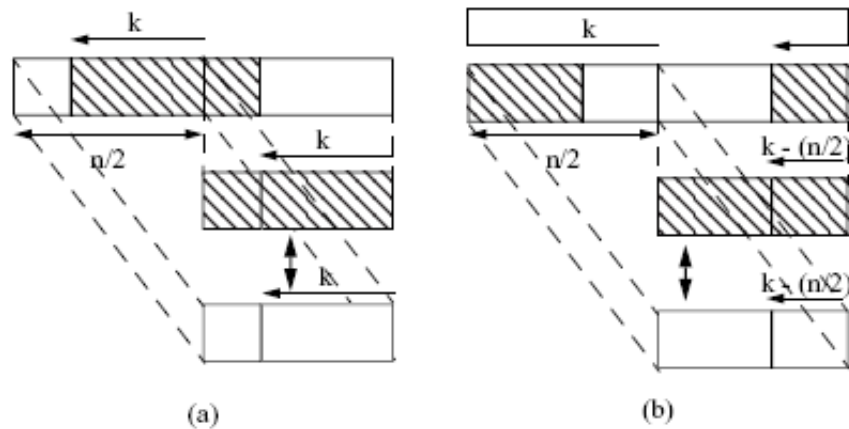
Cyclic Shift on Hypercubes



- Data to be shifted are stored on $n = 2^d$ nodes of d -D hypercube. Order nodes, e.g. 000, 001, 010, ... 111
- To cyclic shift by k places when $k \leq 2^{d-1}$, cyclic shift half of the words by k places on $(d-1)$ -D hypercubes. Then swap first k positions across highest dimen., as shown. Apply algorithm recursively on $(d-1)$ -D hypercubes.



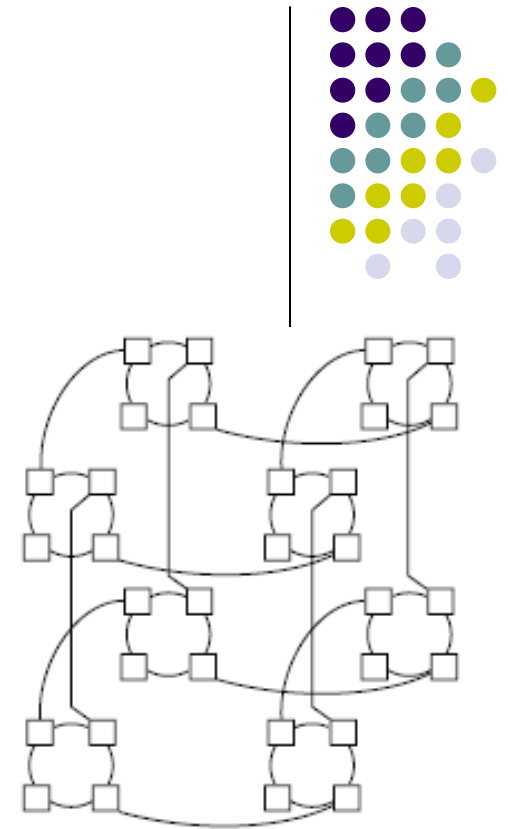
Cyclic Shift on Hypercubes

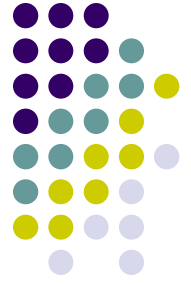


- To cyclic shift by k places when $k > 2^{d-1}$ cyclic shift half words by $k - (2^{d-1})$ places and then swap top positions $2^{d-1} - k$ positions, as shown. Apply algorithm recursively on $(d-1)$ -D hypercubes.

Cube-Connected Cycles

- Each vertex of d -D hypercube has degree d . To reduce the degree, replace each vertex by a cycle of $\geq d$ vertices. Assign one vertex on each corner to each of its d edges.
- Connect the vertex assigned to a particular edge to the vertex on the opposite corner assigned to that edge. In above example, each cycle has 4 vertices, although 3 suffice.

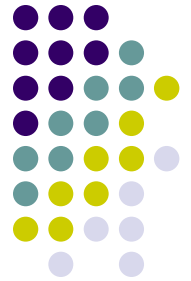




Cube-Connected Cycles

- To swap across dimensions, move data around cycles replacing corners.
- The CCC is useful because ascending and descending normal algorithms can be implemented with only a small loss in time but the CCC can be laid out on the plane much more efficiently than the hypercube. (See book for details.)

Mapping Normal Algorithms to Meshes



- Normal algorithms work well on hypercubes
- Because hypercubes are costly, we map normal algorithms to 1-D and 2-D arrays, which are less so.
- Shuffle and unshuffle operations are key to the translation of normal algorithms to meshes.
- **A shuffle operation on a deck of card:** split the deck in half and interlace the two sets of cards.
- Example on $8 = 2^3$ elements:

Original 0 1 2 3 4 5 6 7

Shuffled 0 4 1 5 2 6 3 7

Shuffle Operations on Linear Arrays



- Consider shuffle of $n = 2^d$ items on 1-D arrays. Can be done by a sequence of swaps of adjacent elements on the array, as shown.

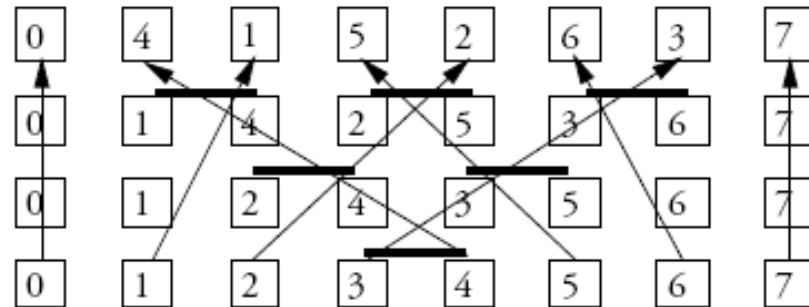
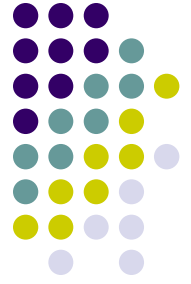


Figure 7.17 The shuffle permutation can be realized by a series of swaps of the contents of cells. The cells between which swaps are done have a heavy bar above them. The result of swapping cells of one row is shown in the next higher row, so that the top row contains the result of shuffling the bottom row.

Shuffle Operations on Linear Arrays



- Could you write a small program for each array processor using the integer it contains as well as the cell number to decide when to swap and when to terminate?
- Number of steps: $n - 1$ where $n = 2^{d-1}$
- **Unshuffle**: reverse shuffle steps

Shuffle Operations and Hypercube Adjacency



Original 0 1 2 3 4 5 6 7

Shuffled 0 4 1 5 2 6 3 7

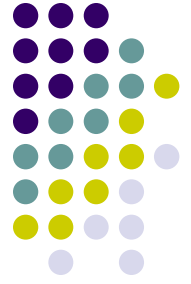
- Represent the eight integers in binary

Original 000 001 010 011 100 101 110 111

Shuffled 000 100 001 101 010 110 011 111

- In original, elements i and $i+1$ for i even differ in the least significant bit. When shuffled, they differ in most significant bit.
- Swapping linearly adjacent elements simulates hypercube swapping.

Ascending Normal Algorithm Simulated on a Linear Array



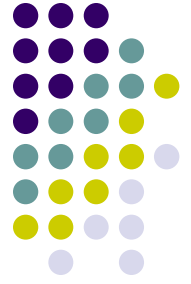
- A normal algorithm swaps values in processors whose indices differ in one bit.

000 001 010 011 100 101 110 111

- Implement swap across first dimension by
 - Swap between even-odd neighbors.
- Implement swap across second dimension by
 - Unshuffle; shuffle blocks of four elements
 - Swap between even-odd neighbors; unshuffle.

000 010 001 011 100 110 101 111

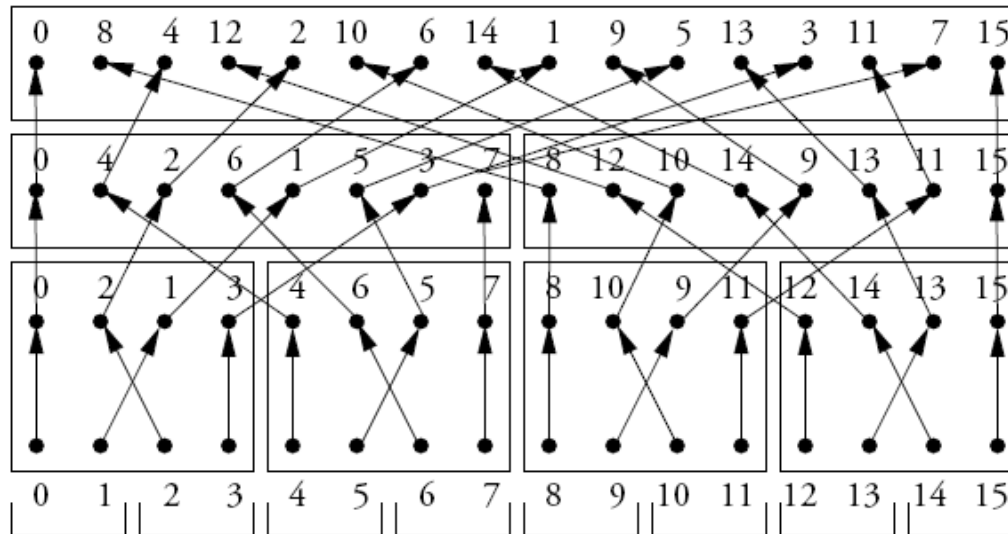
Ascending Normal Algorithm Simulated on a Linear Array



- Implement swap across third dimension by
 - Unshuffle; shuffle blocks of eight elements
 - Swap between even-odd neighbors

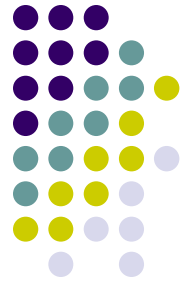
000 100 001 101 010 110 011 111

Normal Ascending Algorithm on Linear Array



- Running time on dD hypercube
 - Shuffles: $2(2^{k-1}-1)$ for $k = 2, 3, \dots, d$.
 - Total $2(2^d - d - 1)$
 - Swaps: d

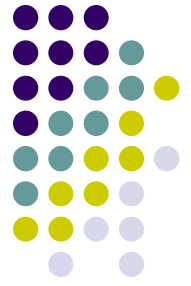
Ascending Normal Algorithms on a Two-Dimensional Array



- Consider $m \times m$ array $\{(r,c)\}$ in row major order

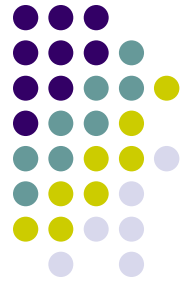
(0,0) (0,1) (0,2) (0,3) (0,4) (0,5) (0,6) (0,7)
(1,0) (1,1) (1,2) (1,3) (1,4) (1,5) (1,6) (1,7)
(2,0) (2,1) (2,2) (2,3) (2,4) (2,5) (2,6) (2,7)
(3,0) (3,1) (3,2) (3,3) (3,4) (3,5) (3,6) (3,7)
(4,0) (4,1) (4,2) (4,3) (4,4) (4,5) (4,6) (4,7)
(5,0) (5,1) (5,2) (5,3) (5,4) (5,5) (5,6) (5,7)
(6,0) (6,1) (6,2) (6,3) (6,4) (6,5) (6,6) (6,7)
(7,0) (7,1) (7,2) (7,3) (7,4) (7,5) (7,6) (7,7)

Ascending Normal Algorithms on a Two-Dimensional Array

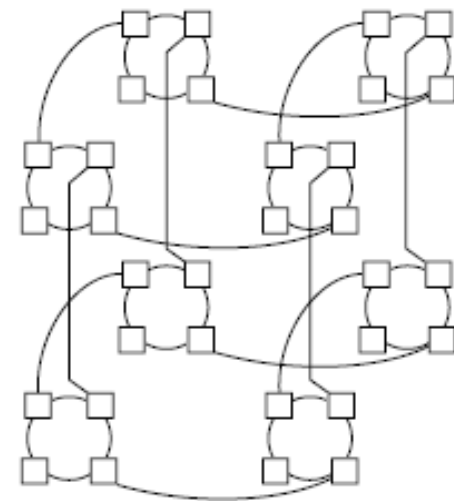


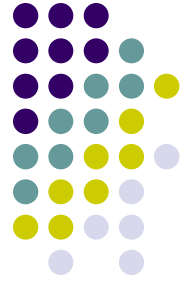
- Treat an index as $cm+r$. All normal alg. exchanges can be done by doing shuffles on rows followed by shuffles on columns.
- Map $n = 2^{2d}$ vertices onto $2^d \times 2^d$ array.
 - Elements (r,c) and $(r,c+1)$ agree in d msb's
 - Elements $(r+1,c)$ and (r,c) agree in d lsb's.
- Ascending algorithm uses shuffles on rows followed by shuffles on columns.
- Algorithm uses $O(\sqrt{n})$ steps, $n=m^2$.

Normal Algorithms on Cube-Connected Cycles



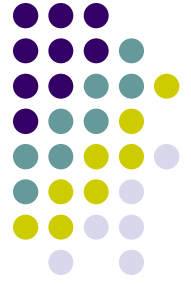
- Consider CCC simulating d-dimensional hypercube
- Let each ring have length $d \leq 2^k \leq 2d$.
- Simulate msbits of ascending algorithm on rings.
- Rotate and swap on ring to simulate asc. alg. on lsbs.
- Running time is $O(d)$.





Routing on Networks

- Data movement on networks is challenging:
 - Contention introduces delay
- Permutation routing: each input maps to unique output
- Local routing network: messages have destination address. Network switches use only this information to route messages.
- A sorting network, such as Batcher's bitonic sorter (Sect. 6.8.1) is a local permutation-routing network.

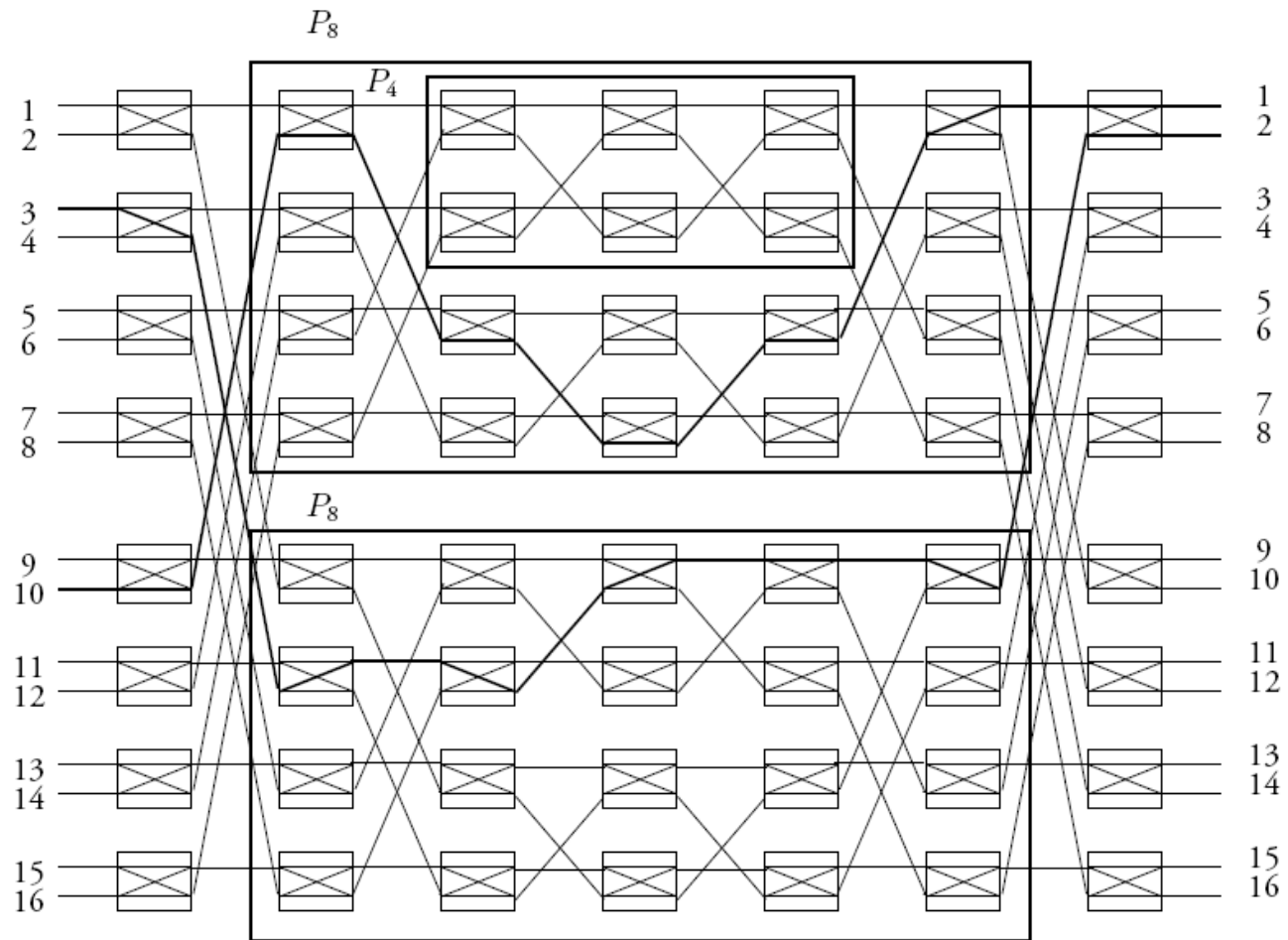


Global Routing Networks

- Global routing networks have knowledge of destinations for each message.
- Global permutation routing networks have unique destination for each message.
 - Permutations determined by switch positions.
- Benes permutation network P_n on $n = 2^k$ inputs is formed by placing two n -input *FFT* graphs back-to-back and replacing nodes with switches.



Benes Permutation Network





Benes Permutation Network

- The Benes network P_2 computes the two permutations on two inputs.
- Use induction to show that P_n computes all permutations on n inputs. Let $m = 2^k$. Let permutation $\pi = (\pi(1), \pi(2), \dots, \pi(m))$ and $\pi(1) = a$. Let a be arbitrarily assigned to 2nd copy of $P_{m/2}$. Let a appear at output $\lceil a/2 \rceil$ and at output a of the $\lceil a/2 \rceil^{\text{th}}$ switch. Determine the other input to this switch. Route it via 1st copy of $P_{m/2}$. This constrains inputs. Bounce back and forth until can't find another input to route. Pick an arbitrary other input to route.