Overview

- Mesh-based architectures
- Hypercubes
- Embedding meshes in hypercubes
- Normal algorithms on hypercubes
- Summing and broadcasting on hypercubes
- Cyclic shift on hypercubes
- Cube-connected cycles – an architecture to efficiently embed hypercubes in the plane
Mesh-Based Machines

- d-tuple \( (n_1, n_2, ..., n_d) \) characterizes a d-dimensional mesh. \( n_i \) is number of vertices in the \( i \)th dimension. A \( (n_1, n_2, ..., n_d) \)-mesh is two \( (n_1, n_2, ..., n_{d-1}) \)-meshes with an edge between corresponding vertices.

- It has \( n_1 n_2 ... n_d \) vertices. The 1-D mesh has \( n_1-1 \) edges; the 2-D mesh has \( (n_1-1)(n_2-1) \) edges; and the d-D mesh has \( (n_1-1)(n_2-1)...(n_d-1) \) edges.
Sorting on a 1D Array

- **Bubble sort** compares adjacent elements from right to left and "bubbles up" the largest elements. Its running time is $O(n^2)$.
- **Odd-even transposition sort** compares odd with even elements and then even with odd elements $n-1$ times.
- Correctness shown using 0-1 principle.
Embedding 1D Arrays in 2D Arrays

- Snake-row ordering of 1D array in 2D
  - Running time of 1D algorithm on 2D array unchanged.
  - Implementation of one step of 2D algorithm on 1D array in six steps:
    - Compute locally
    - Exchange data with column neighbors
    - Even (odd) rows send (receive) data from odd (even) rows.
  - Simulation of T-step n×n computation on 1D array takes at most (8n-1)T steps.
Hypercube-Based Machines

- A d-dimensional hypercube has $2^d$ vertices labeled by binary d-tuples. Edges between vertices whose d-tuples differ in one position.
- A d-D hypercube has $(d/2) 2^d$ edges. It can be formed by adding edges between corresponding vertices of two (d-1)-D hypercubes.
Graph Embeddings

- Graph G is **embedded** in graph G’ by mapping each vertex of G with a unique vertex of G’ and mapping edges of G into non-intersecting paths in G’.

- An embedding has a **dilation** of λ if the longest path into which an edge of the original graph is mapped is λ.
Embedding Meshes in Hypercubes

- Embeddings of meshes into 1D, 2D, and 3D hypercubes:

- How is this embedding done?

- In each embedding dilation $\lambda = 1$. 
Embedding Meshes in Hypercubes

**Theorem** If d is even (odd), a \((2^{d/2}, 2^{d/2})\)-mesh (a \((2^{(d+1)/2}, 2^{(d-1)/2})\)-mesh) can be embedded in a d-D hypercube with dilation \(\lambda = 1\).

**Proof** *(Basis)* The \((2,2)\)-mesh & 2-D hypercube are the same. The \((4,2)\)-mesh can be embedded into the 3-D hypercube by cutting two edges on one face.

*(Induction Hypothesis)* Assume such embeddings exist for \(d < N\). We show they exist for \(d = N\).

*(Inductive Step)* \(d\) odd & even.
Embedding Meshes in Hypercubes

d odd
Let k=(d-1)/2 or d = 2k+1. The \((2^{k+1}, 2^k)\)-mesh consists of two \((2^k, 2^k)\)-meshes. By hypothesis, each has embedding in \((d-1)\)-D hypercube. Map to the \((2k+1)\)-D cube as follows:

a) Map each \((2^k, 2^k)\)-mesh to a \(2^{2k}\)-D cube.
b) Place both meshes on the plane, one above the other. Flip the second mesh.

c) Add edges between the two faces. These edges are in the \(2^{2k+1}\)-D cube containing the mesh, \(2k+1 = d\).
Let $k = d/2$. The $(2^k, 2^k)$-mesh can be represented as two $(2^k, 2^{k-1})$-meshes. Each can be embedded into $(2k-1)$-D hypercube as shown. These two resultant meshes are then laid out along their long sides and edges added so it can be embedded in hypercube.
Normal Algorithms

- A normal algorithm on a hypercube is one in which data moves synchronously across one dimension at a time.
- Many algorithms, such as FFT, are normal.

- Input data resides in vertices at level 0. To do computations at level 1, exchanges are done with neighbors differing on least significant bit. Computations at level 2 are done after exchanges between neighbors differing on most significant bit.
Normal Algorithms

- **Ascending** normal algorithms make exchanges across dimensions 0, 1, 2, ...
- **Descending** normal algorithms make exchanges across descending dimensions.
  - Ex: FFT is an ascending normal algorithm
- Batcher’s bitonic sort algorithm (see book – note correction) can also be realized as a normal algorithm on a hypercube.
Summing & Broadcasting on Hypercubes

- **Summing**: A datum is placed on each vertex of a d-dimensional hypercube.
- On 1st cycle, for each tuple \((a_{(d-1)}, ..., a_0)\), vertex \((a_{(d-1)}, ..., a_1, 1)\) sends it data to vertex \((a_{(d-1)}, ..., a_1, 0)\) where it is added.
- On 2nd cycle vertex \((a_{(d-1)}, ..., a_2, 1, 0)\) sends it data to vertex \((a_{(d-1)}, ..., a_2, 0, 0)\) where it is added.
- Repeat until all data is summed at \((0,0,..., 0)\)
- This is an ascending normal alg. Data moves across dimensions in ascending order.
- **Broadcast** by reversing the data transmissions.
Cyclic Shift on Hypercubes

- Data to be shifted are stored on \( n = 2^d \) nodes of a d-D hypercube. Order nodes, e.g. 000, 001, 010, ….
- To cyclic shift by \( k \) places when \( k \leq n/2 \), first shift half words by \( k \) places then swap first \( k \) positions across highest dimension, as shown. If \( n = 2^d \), do this work recursively on a d-D hypercube.
Cyclic Shift on Hypercubes

- To cyclic shift by $k$ places when $k > n/2$, cyclic shift half words by $k-(n/2)$ places and then swap top positions $n-k$ positions, as shown. If $n = 2^d$, do this work recursively on a $d$-D hypercube.
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.)