Processor Topology

- Describes the “scheme” or “structure” by which different processors are connected
- Influences the method and cost of communication between nodes
- Considerations include sharing of physical links, congestion, distance between nodes
- (Quite different from mathematical notion of topology, though related)

Characterizations

- Using some tools from graph theory…
- Topologies are most generally represented by a *graph*, where processors are nodes, and edges represent direct connectivity (via ethernet or other hardware)
- Typically want (and assume) that all processors have the same edge degree in the graph

Characterizations

- For a network topology graph G, define its diameter, d(G), as the longest shortest path between any two nodes in the graph.
  - i.e. the worst case communication cost between nodes, assuming routing is done intelligently
- Exercise: can you put concrete bounds on the diameter of a graph in terms of the number of nodes and edges? How about vice versa?

Characterizations

- Messages competing for delivery to a node may cause network congestion, or contention
- Define the bisection width as the smallest number of edges which, when removed, would partition the network into two connected graphs
  - Ex. for a linear array, the b.w. is 1
- A bisection width of w guarantees that w messages can be in transit simultaneously in the network
  - However, one can typically have more in special cases

Bisection and Bandwith

- Bisection width also gives a quantification of redundancy; if there are failures in the communication hardware, we know how many alternate routes are still available
- However, we also need to include a measure of how much data can be sent across each connection
- The bisection bandwidth is thus the product of the bisection width and the bandwidth of the wires

Linear arrays and rings

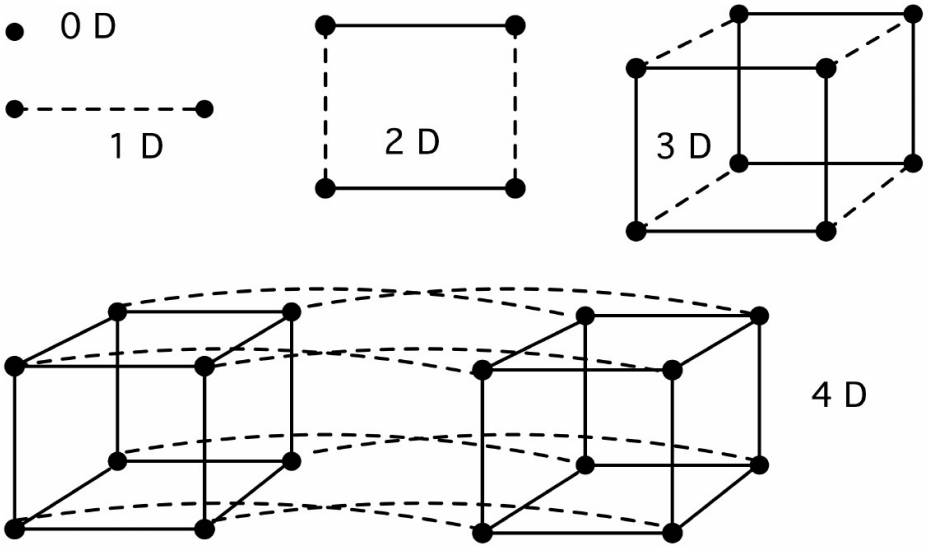
- Linear array: connect a processor P_i to both P_{i+1} and P_{i-1}
  - In graph theory, this is a Path Graph
- We can optionally connect the first to the last processor to get a ring configuration
  - In graph theory, known as a Cycle Graph
- If our network is a linear array, what are the implications for the cost of collectives such as broadcasting?

Mesh arrangements

- Processors can also be arranged into a grid-like coordinate system
  - In addition to rank, each processor will also get coordinates, e.g. (i,j,k) for a 3-D mesh
- Optionally, we can transform the mesh into a “torus” by connecting edge nodes to opposite edge nodes
- Exercise: what is the bandwidth, diameter, of grid arrangements? Torus arrangements?

Hypercubes

- Used to get good “nearest-neighbor” behavior, with a smaller diameter than a mesh
- Don’t want a fully-connected network
- Formed by 2^n processors
- Easily described by bit sequences



Exercise: What is the diameter? Bisection width?

Hypercubes

- Can “embed” a 1-D mesh into an N-D hypercube using Gray Codes

1D Gray code:			0	1					
2D Gray code:	1D code and reflection:	0	1	⋮	1	0			
	append 0 and 1 bit:	0	0	⋮	1	1			
3D Gray code:	2D code and reflection:	0	1	1	0	⋮	0	1	1	0
		0	0	1	1	⋮	1	1	0	0
	append 0 and 1 bit:	0	0	0	0	⋮	1	1	1	1

- It follows that we can also embed higher-dim meshes! (How?)

Declaring Topologies in MPI

- “Virtually” defined topologies
  - MPI_UNDEFINED
  - MPI_CART
  - MPI_GRAPH
  - MPI_DIST_GRAPH
- Use MPI_Topo_test(MPI_Comm, int*) to check which one your communicator is using

MPI Grids

- Create a grid:

```
int MPI_Cart_create(
    MPI_Comm comm_old, // original comm
    int ndims, // dimension of the grid
    int *dims, // # of coords in each axis
    int *periods, // whether each is periodic
    int reorder, // whether rank can change
    MPI_Comm *comm_cart  // new comm
)
```

- MPI_Cart_coords(MPI_Comm, int, int, int*)
- MPI_Cart_rank(MPI_Comm, int*, int*)

Grid example

```
// cart.c
MPI_Comm comm2d;
ndim = 2; periodic[0] = periodic[1] = 0;
dimensions[0] = idim; dimensions[1] = jdim;
MPI_Cart_create(comm,ndim,dimensions,periodic,1,&comm2d);
MPI_Cart_coords(comm2d,procno,ndim,coord_2d);
MPI_Cart_rank(comm2d,coord_2d,&rank_2d);
printf("I am %d: (%d,%d); originally %d\n",rank_2d,coord_2d[0],coord_2d[1],
    procno);
```

Note:

- The two different communicators
- Periodicity is axis-by-axis
- The rank in the old communicator may change!
  - This may happen due to network and hardware constraints, to better represent the grid structure
- Inside the grid comm, if we specify a processor coordinates outside the grid, will be regarded as MPI_PROC_NULL!
- Typically will want non-blocking communication
  - MPI_Isend, MPI_Irecv

General Graph Topology

- Can construct arbitrary graph structure between nodes
  - MPI_Dist_graph_create
- Can also specify “weight” of edges between nodes
  - Maybe communication is asymmetric!
- Minimal description:
  - Number of neighbors for each node
  - Ranks of the neighbors

Topology-based Collectives

- MPI_Neighbor_allgather,
  MPI_Neighbor_alltoall, with standard variants
- MPI_Dist_graph_neighbors_count
  MPI_Dist_graph_neighbors_count