Intro to MPI
Last Time ...

Intro to Parallel Algorithms
- Parallel Search
- Parallel Sorting
  - Merge sort
  - Sample sort
Today

- Network Topology
- Communication Primitives
  - Message Passing Interface (MPI)
- Randomized Algorithms
  - Graph Coloring
Network Model

- Message passing model
- Distributed memory nodes
- Graph $\mathcal{G} = (N, E)$
  - nodes $\rightarrow$ processors
  - edges $\rightarrow$ two-way communication link
- No shared RAM
- Asynchronous
- Two basic communication constructs
  - `send (data, to_proc_i)`
  - `recv (data, from_proc_j)`
Network Topologies

- **Line**
- **Ring**
- **Mesh**
- **Wraparound mesh**

**Link:** one task pair, bidirectional

**Send-Recv time:**

\[ \text{latency} + \text{message/bw} \]

**Node:** one send/recv

**Hypercube?**
**Hypercube**

(a) Binary 1-cube, built of two binary 0-cubes, labeled 0 and 1

(b) Binary 2-cube, built of two binary 1-cubes, labeled 0 and 1

(c) Binary 3-cube, built of two binary 2-cubes, labeled 0 and 1

(d) Binary 4-cube, built of two binary 3-cubes, labeled 0 and 1

**generic term**

3-cube, 4-cube, \ldots, q-cube

d dimensions, \(2^d\) processes

two processes connected iff they differ in only one bit

distance = \(\log p\)
Network topologies

Fat Tree
Basic concepts in networks

- Routing (delivering messages)
- Diameter
  - maximum distance between nodes
- Communication costs
  - Latency: time to initiate communication
  - Bandwidth: channel capacity (data/time)
    - Number and size of messages matters
- Point-to-point and collective communications
  - Synchronization, all-to-all messages
Network Models

Allow for thinking and measuring communication costs and non-locality
p2p cost in MPI

- **start-up time**: $t_s$
  - add header/trailer, error correction, execute the routing algorithm, establish the connection between source and destination

- **per-hop time**: $t_h$
  - Time to travel between two directly connected nodes
    - *node latency*

- **per-word transfer time**: $t_w$

We will assume that the cost of sending a message of size $m$ is:

$$t_{comm} = t_s + t_w m$$
Network Metrics

- **Diameter**
  - Max distance between any two nodes

- **Connectivity**
  - Number of links needed to remove a node

- **Bisection width**
  - Number of links to break network into equal halves

- **Cost**
  - Total number of links

- **Ring, Fully connected ring**
  - **Diameter**
    - $\frac{p}{2}$
  - **Connectivity**
    - $2$
  - **Bisection Width**
    - $2$
  - **Cost**
    - $\frac{p(p - 1)}{2}$
Network Metrics

- **Diameter**
  - Max distance between any two nodes
- **Connectivity**
  - Number of links needed to remove a node
- **Bisection width**
  - Number of links to break network into equal halves
- **Cost**
  - Total number of links

- **Hypercube**
  - Diameter $\log p$
  - Connectivity $\log p (= d)$
  - Bisection Width $p/2$
  - Cost $\frac{p \log p}{2}$
Example problem

- how would you perform a fully distributed matrix multiplication (dgemm)
  
  - Assume a 2D grid topology
  - Need to compute $C = A \times B$
    
    $$C_{ij} = \sum_k A_{ik} B_{kj}$$
  
  - How to divide data across processors?
  - What is the local work?
  - Communication? Costs?
Matrix-Matrix Multiplication on 2D Mesh

- Compute $C = AB$, $p = n \times n$
- Rows of $A$ fed from left
- Columns of $B$ fed from right
- Operation is synchronous
- $ij$ indexing of processor ids
- Each $ij$ processor computes $C_{ij}$
Matrix-Matrix Multiplication

% i,j process id
% p=n, 1 element per process
for k=1:n
    recv ( A(i, k), (i-1, j) )
    recv ( B(k, j), (i, j-1) )

    C(i,j) += A(i,k)*B(k,j);

    send ( A(i,k), (i+1, j) )
    send ( B(k,j), (i, j+1) )

A(1,4) A(1,3) A(1,2) A(1,1)
A(2,4) A(2,3) A(2,2) A(2,1)
A(3,4) A(3,3) A(3,2) A(3,1)
A(4,4) A(4,3) A(4,2) A(4,1)
Collective Communications
MPI Collectives

- Communicator (MPI_Comm)
  - determines the scope within which a point-to-point or collective operation is to operate
  - Communicators are dynamic
    - they can be created and destroyed during program execution.
    - MPI_COMM_WORLD

- int MPI_Barrier(MPI_Comm comm); avoid
  - Synchronize all processes within a communicator
Data Movement

- broadcast
- gather(v)
- scatter(v)
- allgather(v)
- alltoall(v)
int MPI_Bcast(void* buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm)
Gather & Scatter

```
int MPI_Gather(void* sbuf, int scount, MPI_Datatype stype,
               void* rbuf, int rcount, MPI_Datatype rtype,
               int root, MPI_Comm comm)
```

```
int MPI_Scatter(void* sbuf, int scount, MPI_Datatype stype,
                void* rbuf, int rcount, MPI_Datatype rtype,
                int root, MPI_Comm comm)
```
Gather & Scatter

int MPI_Gather(void* sbuf, int scount, MPI_Datatype stype,
               void* rbuf, int rcount, MPI_Datatype rtype,
               int root, MPI_Comm comm )

int MPI_Scatter(void* sbuf, int scount, MPI_Datatype stype,
                void* rbuf, int rcount, MPI_Datatype rtype,
                int root, MPI_Comm comm)
int MPI_Allgather(void* sbuf, int scount, MPI_Datatype stype, 
void* rbuf, int rcount, MPI_Datatype rtype, MPI_Comm comm)
int MPI_Alltoall(void* sbuf, int scount, MPI_Datatype stype,
    void* rbuf, int rcount, MPI_Datatype rtype, MPI_Comm comm)
Global Computation

- Reduce
- Scan
Reduce

int MPI_Reduce(void* sbuf, void* rbuf, int count, MPI_Datatype stype, MPI_Op op, int root, MPI_Comm comm)

int MPI_Allreduce(void* sbuf, void* rbuf, int count, MPI_Datatype stype, MPI_Op op, MPI_Comm comm)

int MPI_Reduce_scatter(void* sbuf, void* rbuf, int* rcounts, MPI_Datatype stype, MPI_Op op, MPI_Comm comm)
Scan (prefix-op)

int MPI_Scan(void* sbuf, void* rbuf, int count,
             MPI_Datatype datatype, MPI_Op op, MPI_Comm comm)
Assignment 1 – Problem 1

Implement samplesort in parallel

- Samplesort

1. Local sort, pick samples
2. Gather samples at root
3. Sort samples, pick splitters
4. Broadcast splitters
5. Bin data into buckets
6. AlltoAllv data
7. Local sort