COMMUNICATION OPERATIONS AND MESSAGE PASSING

- Introduction to programming with message passing
- Broadcast operations
- All-to-all broadcast and reduction operations
- Scatter and Gather operations
- All-to-all personalized communication

Introduction to message-passing

- Need to explicitly code the exchange of messages [data, control,..]

Example:
Revisit the sum example seen earlier

**ALGORITHM : 1.** Parallel Sum of \( n \) numbers
1. for \((j = 0; j < p; j++)\) \{ // Parallel Loop
2. \( \text{tmp}(j) = 0; \)
3. for \((i = j * m; i < (j + 1) * m; i++)\)
4. \( \text{tmp}(j) = \text{tmp}(j) + x(i) \); \}
5. \( s = 0; \)
6. for \((j = 0; j < p; j++)\) // Sequential loop
7. \( s = s + \text{tmp}(j); \)

Let “root” = ‘master’ node where the sum ends up. Recall: \( m = n/p \)

**ALGORITHM : 2.** Parallel Sum with communication
1. If \((\text{myid} == \text{root})\) \{}
2. \( \text{read array } x; \)
3. \( \text{For } (j = 0, j < p \& \& j! = \text{root}; j++) \)
4. \( \text{send } x(j * m : (j + 1) * m - 1) \text{ to proc. } j \) \}
5. else
6. \( \text{receive } x\text{loc}(1 : m) \text{ from root } ; \)
7. \( \text{tmp} = 0; \)
8. for \((i = 0; i < m; i++)\)
9. \( \text{tmp} += x\text{loc}(i); \)
10. \( \text{REDUCE(sum, tmp,'ADD')} \) Reduction oper.

REDUCE\((\text{sum, tmp,'ADD'})\) adds ‘tmp’ from each PE into ‘sum’
- Can do reductions with add, multiply max, min, etc..
- More on reductions later.
- Next: we will see some of the common communication functions used –
**Communication 'kernels'**

**Typical questions addressed:**

1. Identify the important communication operations
2. Find effective algorithms for performing these on distributed memory computers
3. Analyze their cost
   - A by-product: some framework for generic algorithms

---

**Example: Broadcast operation**

- Sending a message from a 'root' node to all nodes is a Broadcast operation.

**Questions:** Best way to broadcast a message from a root node to all others in a ring? In hypercube?

---

**Standard broadcast and reduction operations**

- Reduction does a global operation (e.g. a sum) on items located on all processors onto a 'root' processor
- Can be viewed as a sort of inverse of the broadcast

In parallel sum example, could replace the sends of \( x(j \cdot m : (j + 1) \cdot m - 1) \) from root to all others by a broadcast of all \( x \) from root of the vector \( x \). Lines 1 – 6 replaced by:

1. broadcast(x,root)

- Note however that each PE will get the whole vector.
- Corresponding MPI code provided in class web-site.
**All-to-all broadcast and reduction**

- All-to-all broadcast can be viewed as $p$ broadcasts, one from each node.
- Similarly: All-to-all reduction is a reduction to each node (different for each node).

**Note:** All-reduce ($\neq$ all-to-all reduce) is a reduction operation in which the result of reduction is available in each processor

- All-reduce achievable by a reduce followed by a broadcast [not best way]

- Important application of all-reduce: testing if an algorithm has "converged".

**Example:** Test would be something like:

```plaintext
if $\max_{i} |x^i_k - x^i_{k+1}| < \text{then stop}$
```

- Variable $i = \text{processor}$, variable $k = \text{iteration number}$
- Need to know $\max_{i} |x^i_k - x^i_{k+1}|$ in each processor.
- See text for algorithms on linear array, ring, and hypercubes

---

**Gather and scatter operations**

- Scatter is similar to a broadcast – but a different item is sent to each processor -
- Gather does the inverse operation.

**Question:** How would you implement a Scatter operation on a hypercube?  
**Cost?**
For the parallel sum example – we can “scatter” the subvectors to be summed up in each processors.

In parallel sum algorithm, the lines

```c
for (j = 0; j < p & & j != root; j++) {
    send x(j*m : (j+1)*m-1) to process j
else
    receive xloc(1:m) from root ;
}
```

are replaced by

3. scatter(x)

---

**All-to-All personalized communication**

Can be viewed as a scatter from each node: each node sends a distinct message to every other node.

```
P_0 \begin{bmatrix} A_0 & A_1 & A_2 & A_3 \end{bmatrix} \rightarrow P_0 \begin{bmatrix} A_0 & B_0 & C_0 & D_0 \end{bmatrix}
P_1 \begin{bmatrix} B_0 & B_1 & B_2 & B_3 \end{bmatrix} \rightarrow P_1 \begin{bmatrix} A_1 & B_1 & C_1 & D_1 \end{bmatrix}
P_2 \begin{bmatrix} C_0 & C_1 & C_2 & C_3 \end{bmatrix} \rightarrow P_2 \begin{bmatrix} A_2 & B_2 & C_2 & D_2 \end{bmatrix}
P_3 \begin{bmatrix} D_0 & D_1 & D_2 & D_3 \end{bmatrix} \rightarrow P_3 \begin{bmatrix} A_3 & B_3 & C_3 & D_3 \end{bmatrix}
```

- Equivalent to $p$ gathers too (one to each node)
- Notice: operation amounts to transposing a $p \times p$ array!

---

How would you code an all-to-all communication on a hyper-cube?