Unit 3: Parallel Communication

One to All Broadcast and all to one reduction

1. Parallel algorithms require a single process to send identical data to all other processes or a subset of processes. This is **one to all broadcast.**
2. In all to one reduction, the data from all process are combined through an **associative operator** and are accumulated at a single destination process into 1 buffer size m.
3. Reduction is used to find the sum, product, or minimum / maximum of set of numbers.

Ring or Linear Array :

1. Recursive doubling → Source process first sends a message to another process, both these processes simultaneously send the message to other 2 processes that are still waiting for the message.
2. The message can be broadcasted in log p steps.
3. Reduction on linear array is performed by simply reversing the direction and sequence of communication.
4. Each odd numbered node sends its buffer towards and even numbered node.

Matrix Vector multiplication:

1. All rows of the matrix must be multiplied with the vector, each process needs the vector.
2. Hence before computing the product, each column of nodes performs a one to all broadcast of elements.
3. After broadcast, each process multiplies its matrix element with the result of the broadcast. Each row of processes needs to add its result to generate the corresponding element of the product vector.

Mesh:

1. One to all broadcast on 2 D square mesh with root p rows and columns.
2. One to all broadcast → On the first row .

3. These nodes then do a one to all broadcast in their columns.

Hypercube:

1. Mesh algorithm is extended to hypercube → process is carried out in d steps. where d is the dimension.
2. Communication starts along the highest dimension (contains the most significant bit of the binary representation)
3. Source and destination nodes procedure → similar to the linear process.
4. The broadcasting node (let's call it the root) sends the message to all its neighbors. Each neighbor then relays the message to its own neighbors, excluding the one from which it received the message. This process continues until all nodes have received the message.
5. However, the outcome isnt affected with which dimensions are chose for communication.
6. No congestion would be faced.

Balanced Binary tree:

1. Leaf node → Processing node
2. Intermediate nodes → switching units

Cost analysis:

p processes → log p steps

Total time t = (ts + tw*m) log p

All to all broadcast and reduction

All to all broadcast → Generalisation of one to all broadcast in which p nodes simultaneously initiate a broadcast.

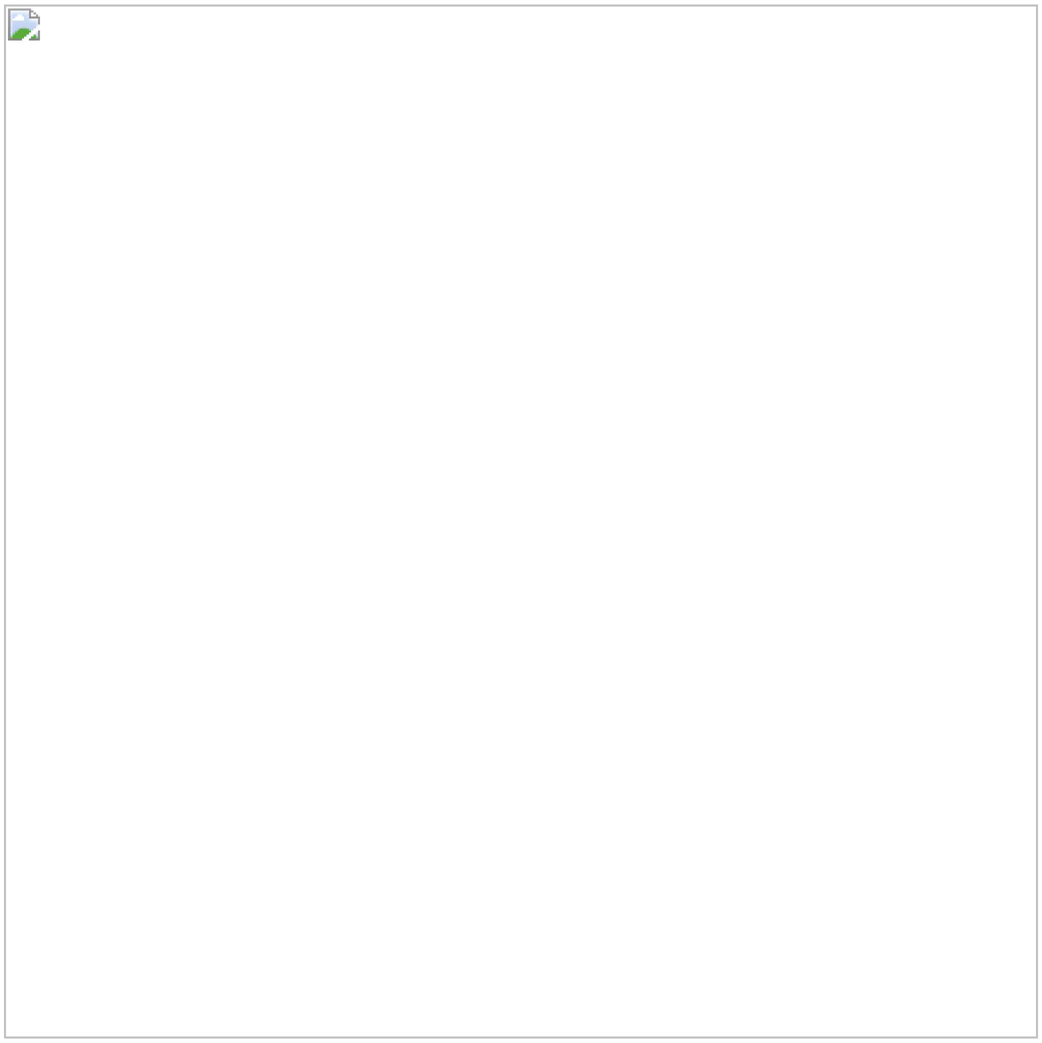
Each processor is the source and the destination

Each process sends a message to all other processes in the system.

Different processes broadcast different messages.

All to all broadcast → Matrix multiplication and matrix vector multiplication.

All to all reduction → every node is the destination of all all to one reduction., processes combine the received message using reduction operation.

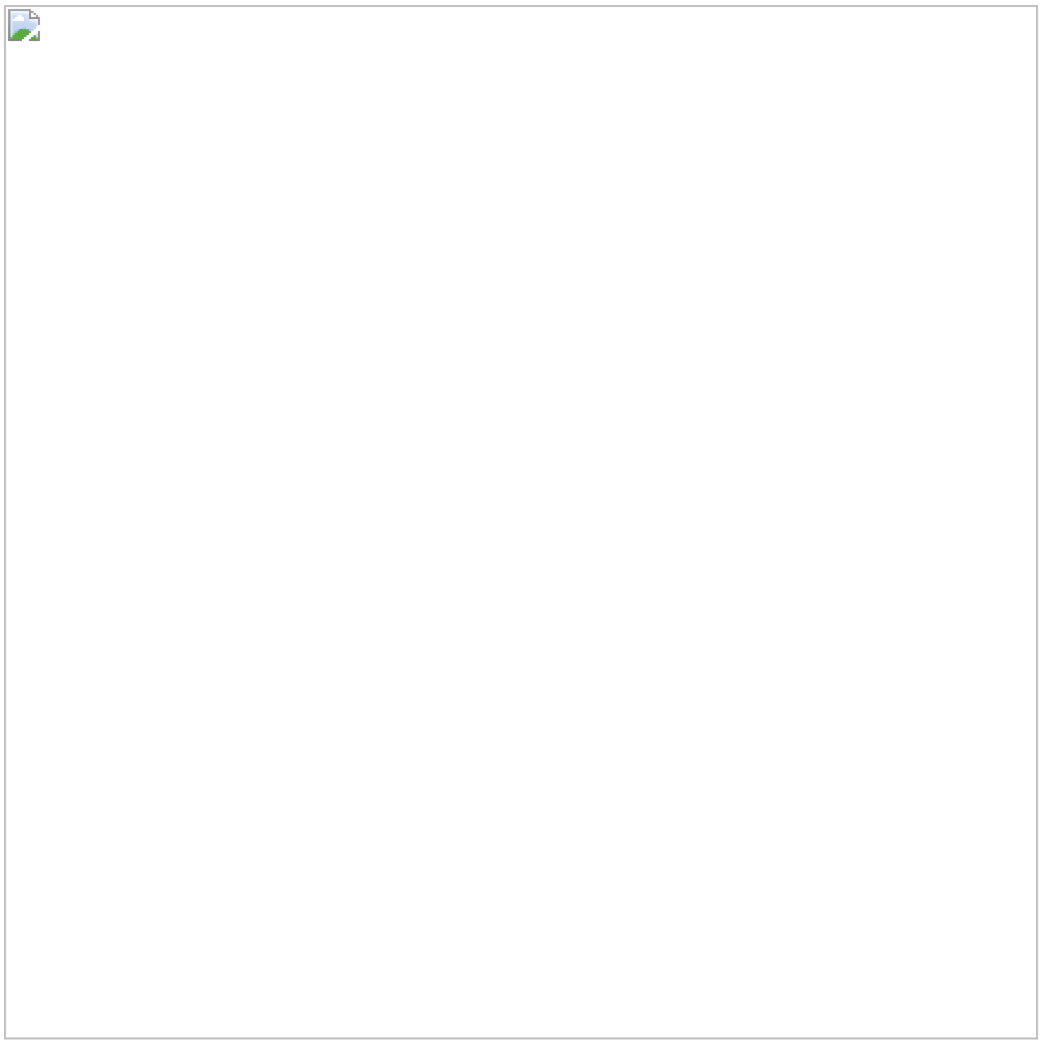


All to all broadcasts → p one to all broadcasts

It is possible to use the communication links in the network more efficiently so that all messages traversing the same path are concatenated in a single message whose size is the sum of the sizes of the individual messages.

Topologies:

1. Linear array and ring



for i to p-1 times, send the message to the right and recieve the message to the left.

ALL NODES SHOULD RECIEVE MESSAGES FROM ALL NODES. So the repetition.

2. Mesh

- 2 phases:

- Each row of the mesh performs an all to all broadcast using the linear array procedure.
- Nodes collect all root (p) messages corresponding to root(p) nodes.
- Second phase, all to all broadcast of messages, column wise. Each node obtains the pieces of m words data.

3. Hypercube
    a. Extension of mesh algorithm
    b. Requires log p steps
    c. Communication starts from lowest dimensions and then proceeds along successively higher dimensions.
    d. All to all reduction → reversing the order and the direction of messages.

Cost:

time = (ts + tw*m)(p-1)

All reduce and prefix sum operations

Simple explanation for all-reduce operation

Imagine you have a bunch of computers all connected together, and you want each computer to know the total sum of some numbers they all have.

First, you could do it in two steps:

1. Each computer adds up all the numbers it has.
2. Then, each computer tells its sum to all the other computers so everyone knows everyone else's sum.

But that takes time because each computer has to talk to every other computer.

Instead, you can speed things up using a method called "all-reduce" which involves some clever communication.

Here's how it works:

1. Each computer adds up the numbers it has just like before.
2. Now, instead of directly sharing their sum with everyone, they'll use a pattern called "all-to-all broadcast." This means each computer sends its sum to every other computer.
3. But, instead of sending just the sum, each computer also receives the sums from others and adds them to its own sum. So, after each step, every computer has the sum of all the numbers from all the computers so far.
4. They keep repeating this process until every computer has the total sum.

This method is faster because instead of each computer sending its sum to everyone separately, they're exchanging sums with each other simultaneously.

So, by using this method, all the computers can quickly figure out the total sum of all the numbers they have without waiting for each other.

Cost analysis: (ts + tw*m)log p

Prefix - sums

Here, instead of just finding the total sum, we want to find the sum up to each position in the list of numbers.

For example, if you have numbers [3, 1, 4, 0, 2], you want to get [3, 4, 8, 8, 10], where each number is the sum of all the numbers before it, including itself.

To solve this on our network of computers, we use a similar method to before, but with a little twist:

1. Each computer starts with its number and also a "result buffer" to store the partial sums.
2. They communicate with each other in a specific way: a computer only adds the numbers it receives to its result buffer if they come from computers with smaller labels.
3. After each communication step, they update what they'll send out next based on what they've received.
4. They repeat this process until each computer has its correct prefix sum.

So, it's like a more detailed version of finding the total sum, where each computer also keeps track of partial sums along the way. This way, they can efficiently find the prefix sums they need.

Scatter and gather

1. In scatter, a single node sends a **unique** message of size m to every other node. Its also called **one to one PERSONALISED** communication.
2. Unlike one to all broadcast, one to all personalised (scatter) does not involve duplication of data.

3. The opposite of scatter communication is gather communication.
4. Gather → single node collects a unique message from the other node. This is different from all to one reduction as it doesnt involve combination or reduction of data.

For hypercube,

1. Similar to one to all communication, however the size and the contents of the message are different.

2. The source node, node 0 , contains all the messages, and the messages are identified by the labels of the destination nodes.
3. In the first communication step, the source transfers halfs of its message to its neighbors.
4. Subsequently, half of data is transferred to neighbors.
5. Log p communication steps.

All to all personalised communication

1. Each node sends a distinct message of size m to every other node.

2. Equivalent to transposing a 2-D array of data distributed among p processes using 1-D array.
3. All to all personalised communication is also known as **total exchange.**

4. Used in fast fourier transform, parallel sorting

Ring :

Easy explanation:

Imagine you have six computers connected in a line, and you want each computer to share some data with every other computer in a specific way.

Here's how it works:

1. Each computer has several pieces of data to share with the others. They're labeled like this: {source, destination}, where "source" is the computer sending the data and "destination" is the computer receiving it.
2. In the first step, each computer bundles up all the data it wants to send and sends it as one big message to its neighbor.
3. Each computer receives this big message, picks out the data meant for itself, and passes the rest of the data along to the next computer.
4. This process repeats for several steps, each time passing the remaining data to the next computer.
5. After a few steps, each computer has received all the data from every other computer.

Hypercube:

Imagine you have a bunch of computers arranged in a hypercube, which is like a multi-dimensional grid where each computer is a point. We want each computer to share some data with every other computer in a specific way.

1. We'll use the structure of the hypercube to guide the communication. In each step, pairs of nodes exchange data in a different dimension of the hypercube.
2. For example, in a 3D hypercube, communication happens in three steps, each time using a different dimension (x, y, and z).
3. Each node has some data to share with every other node. So, in each step, it sends part of its data (specifically, half of it) to the corresponding nodes in the other half of the hypercube.

4. This process repeats for several steps, with each step involving different pairs of nodes and different dimensions of the hypercube.
5. After several steps (specifically, log p steps, where p is the number of nodes in the hypercube), each node has exchanged its data with every other node.

Cost analysis :

T = ts*log p + tw*m*(p-1)

Circular shift

1. Circular shift is a member of global communication operations known as permutation, which each node sends a packet of m words to a unique node.
2. Node sends a packet of data to node (i+q)mod p (0 ≤ q ≤ p)

1-shift → p0 to p1, p1 to p2,....pn-1 to p0

2-shift → p0→p2, p2→p4,....pn-2→p0

1. **Initial Shift Along Rows**: In the first stage, the entire set of data is shifted simultaneously by (q mod p) steps along the rows. This means each node sends its data packet to the node that is q steps away in a circular manner. For example, if q is greater than p (the number of nodes), it wraps around to the beginning of the ring again, ensuring a circular shift.
2. **Compensatory Column Shift**: During the circular row shifts, some data packets may traverse the wraparound connection from the highest labeled nodes to the lowest labeled nodes of the rows. These packets must compensate for the distance they lost while traversing the backward edge in their respective rows. To achieve this, after the row shifts, any data packets that traversed from the highest to the lowest labeled nodes must shift an additional step forward along the columns.

3. **Final Shift Along Columns**: Finally, after the compensatory column shift (if needed), the entire set of data is shifted by p steps along the columns. This ensures that the circular shift operation is completed.

cost analysis : time = (ts + tw*m)(root(p) + 1)

Improving speeds of communication operations

1. Splitting and routing messages in parts
    a. Splitting larger messages into smaller parts and routing these parts through different paths can utilise the network better.
    b. One to all broadcast, spiltting a single message M, of size m into p parts, so each message size is (m/p).
    c. Now one to all broadcast → M to splitting into m
    d. Sending these messages needs an all to all broadcast of messages of size m/p residing on each node of scatter operation.
    e. All to one reduction → simply reverse the direction and the sequence of communication.
    f. All reduce → (all to one reduction + one to all broadcast)
    g. All reduce can be accomplished by all to all reduction and then gather folllowed by a scatter and then all to all broadcast.

2. All port communication
    a. In parallel architecture, a single node may have multiple communication ports with links to other nodes.
    b. Nodes can recieve data simultaneously either on the same port or on separate ports.
    c. All port communication can permit simultaneous communication on all the channels.
    d. However, challenges and limitations are : complexity and message size . (when message size is large, it can be split across multiple ports and thats

great ! however, when the message size is similar to the granularity it doesnt make much impact)

e. Bandwidth requirement is high and it must surpass atleast a factor of log p by communication bandwidth.