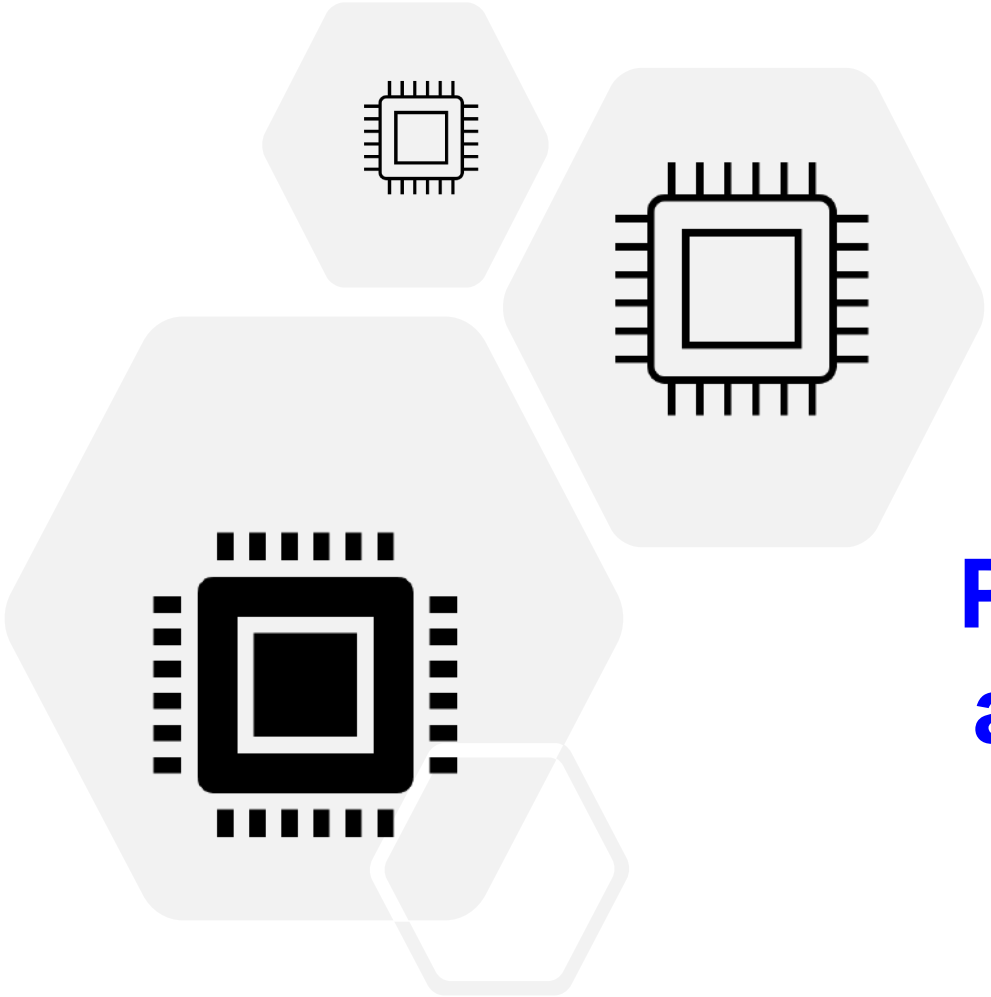
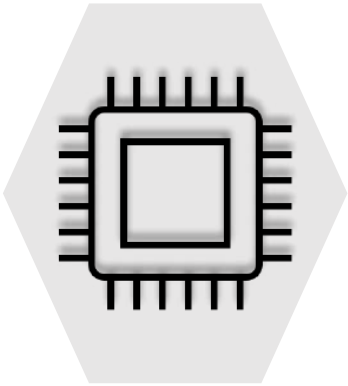
High Performance Computing



Module 3 - Parallel Algorithm and Concurrency

Shafaque Fatma Syed
Assistant Professor - Dept. of Information Technology
A P Shah Institute of Technology, Mumbai

Topics to be discussed



- **Basic Communication Operations**
  - One-to-All Broadcast and All-to-One Reduction
  - All-to-All Broadcast and Reduction
  - All-Reduce and Prefix-Sum Operations
  - Scatter and Gather

Basic Communication Operations: Introduction

- Many interactions in practical parallel programs occur in well-defined patterns involving groups of processors.
- Efficient implementations of these operations can improve performance, reduce development effort and cost, and improve software quality.
- Efficient implementations must leverage underlying architecture. For this reason, we refer to specific architectures here.
- We select a descriptive set of architectures to illustrate the process of algorithm design.

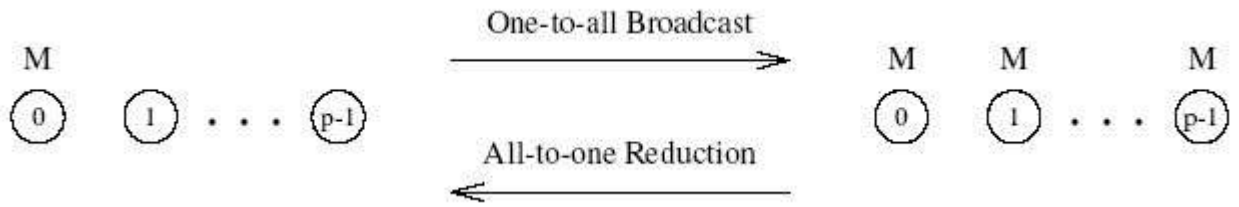
Basic Communication Operations: Introduction

- Group communication operations are built using point-to-point messaging primitives.
- Recall from our discussion of architectures that communicating a message of size m over an uncongested network takes time $t_s + t_m w$.
- We use this as the basis for our analyses. Where necessary, we take congestion into account explicitly by scaling the t_w term.
- We assume that the network is bidirectional and that communication is single-ported.

One-to-All Broadcast and All-to-One Reduction

- One processor has a piece of data (of size *m*) it needs to send to everyone.
- The dual of one-to-all broadcast is *all-to-one reduction*.
- In all-to-one reduction, each processor has *m* units of data. These data items must be combined piece-wise (using some associative operator, such as addition or min), and the result made available at a target processor.

One-to-All Broadcast and All-to-One Reduction

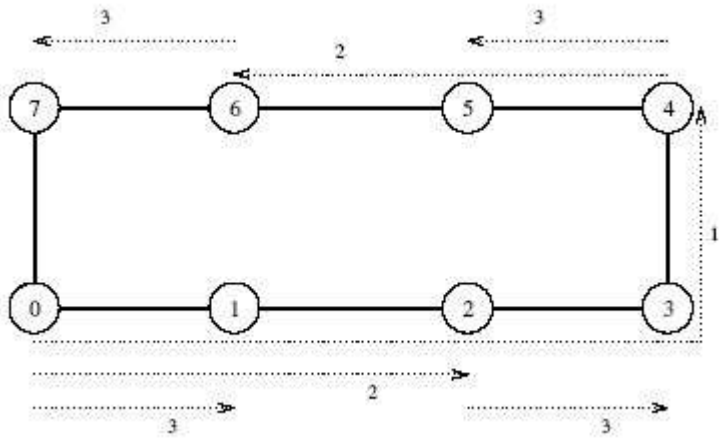


One-to-all broadcast and all-to-one reduction among processors.

One-to-All Broadcast and All-to-One Reduction on Rings

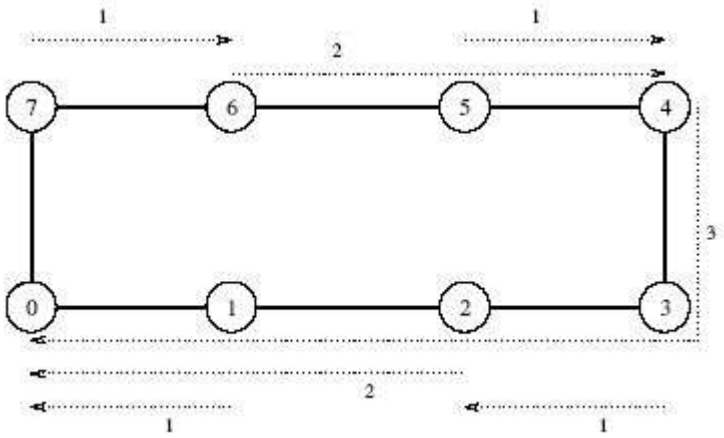
- Simplest way is to send *p-1* messages from the source to the other *p-1* processors - this is not very efficient.
- Use recursive doubling: source sends a message to a selected processor. We now have two independent problems defined over halves of machines.
- Reduction can be performed in an identical fashion by inverting the process.

One-to-All Broadcast



One-to-all broadcast on an eight-node ring. Node 0 is the source of the broadcast. Each message transfer step is shown by a numbered, dotted arrow from the source of the message to its destination. The number on an arrow indicates the time step during which the message is transferred.

All-to-One Reduction



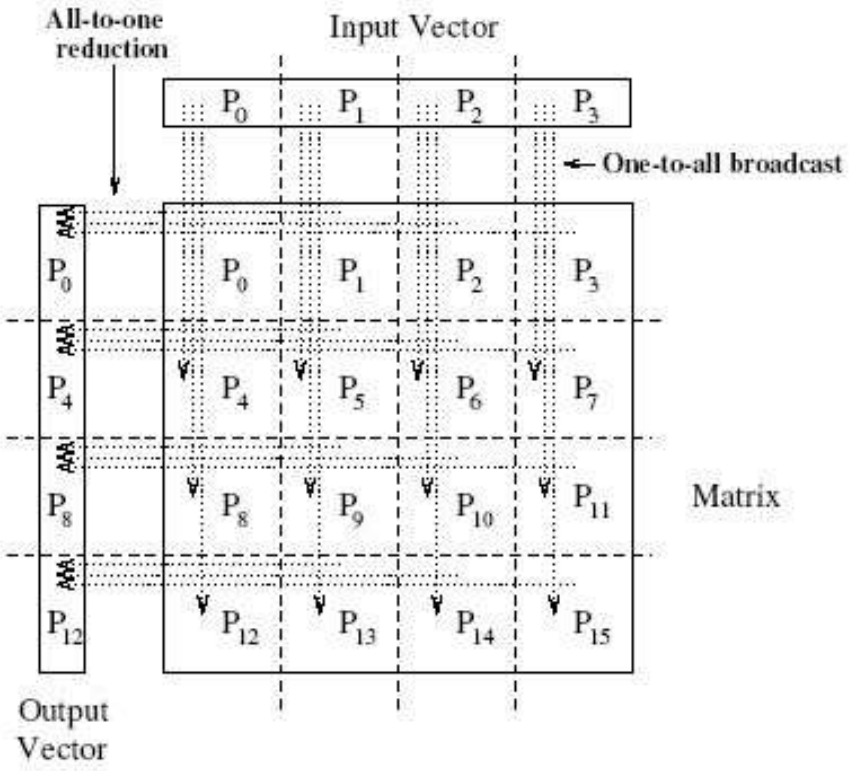
Reduction on an eight-node ring with node 0 as the destination of the reduction.

Broadcast and Reduction: Example

Consider the problem of multiplying a matrix with a vector.

- The *n x n* matrix is assigned to an *n x n* (virtual) processor grid. The vector is assumed to be on the first row of processors.
- The first step of the product requires a one-to-all broadcast of the vector element along the corresponding column of processors. This can be done concurrently for all *n* columns.
- The processors compute local product of the vector element and the local matrix entry.
- In the final step, the results of these products are accumulated to the first row using *n* concurrent all-to-one reduction operations along the columns (using the sum operation).

Broadcast and Reduction: Matrix-Vector Multiplication Example

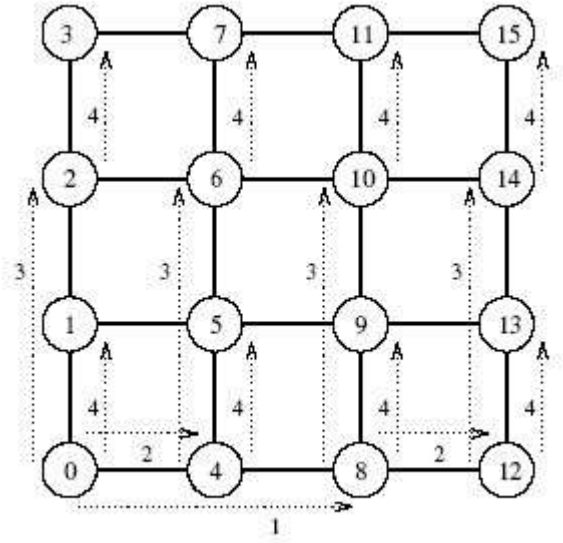


One-to-all broadcast and all-to-one reduction in the multiplication of a *4 x 4* matrix with a *4 x 1* vector.

Broadcast and Reduction on a Mesh

- We can view each row and column of a square mesh of p nodes as a linear array of $\sqrt{p}$ nodes.
- Broadcast and reduction operations can be performed in two steps - the first step does the operation along a row and the second step along each column concurrently.
- This process generalizes to higher dimensions as well.

Broadcast and Reduction on a Mesh: Example

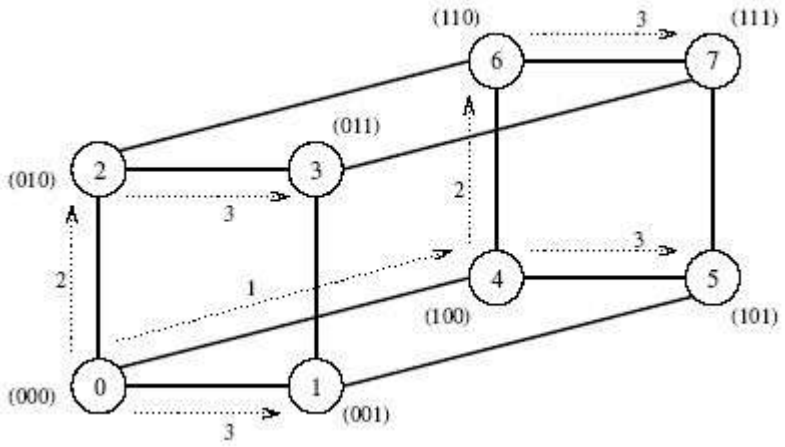


One-to-all broadcast on a 16-node mesh.

Broadcast and Reduction on a Hypercube

- A hypercube with 2^d nodes can be regarded as a d-dimensional mesh with two nodes in each dimension.
- The mesh algorithm can be generalized to a hypercube and the operation is carried out in d (= $\log p$) steps.

Broadcast and Reduction on a Hypercube: Example

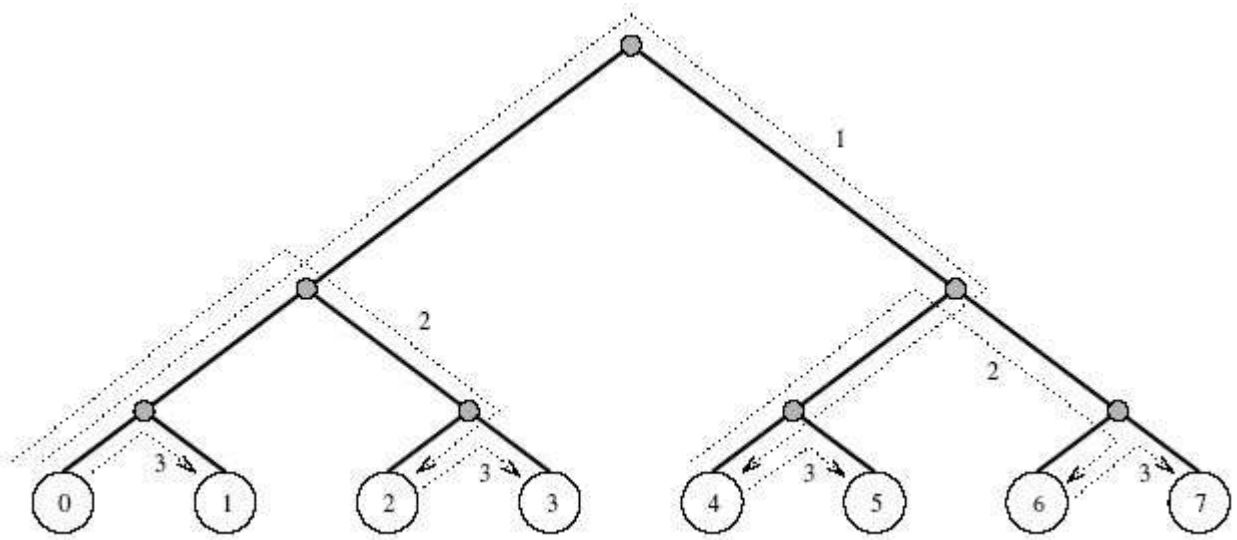


One-to-all broadcast on a three-dimensional hypercube. The binary representations of node labels are shown in parentheses.

Broadcast and Reduction on a Balanced Binary Tree

- Consider a binary tree in which processors are (logically) at the leaves and internal nodes are routing nodes.
- Assume that source processor is the root of this tree. In the first step, the source sends the data to the right child (assuming the source is also the left child). The problem has now been decomposed into two problems with half the number of processors.

Broadcast and Reduction on a Balanced Binary Tree

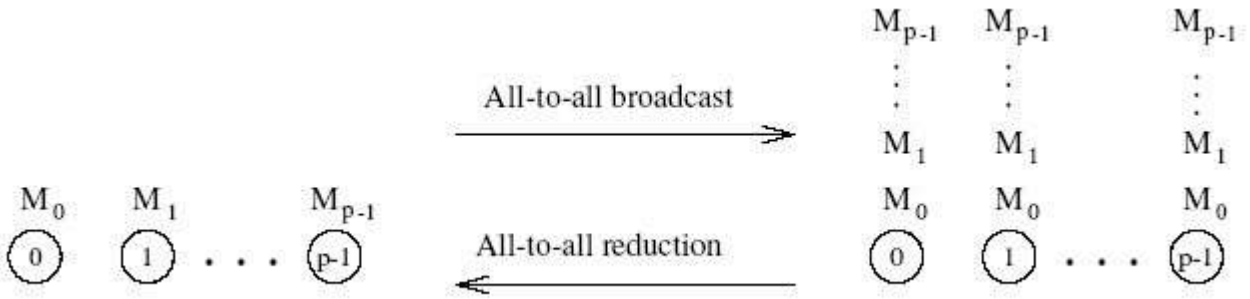


One-to-all broadcast on an eight-node tree.

All-to-All Broadcast and Reduction

- Generalization of broadcast in which each processor is the source as well as destination.
- A process sends the same *m*-word message to every other process, but different processes may broadcast different messages.

All-to-All Broadcast and Reduction

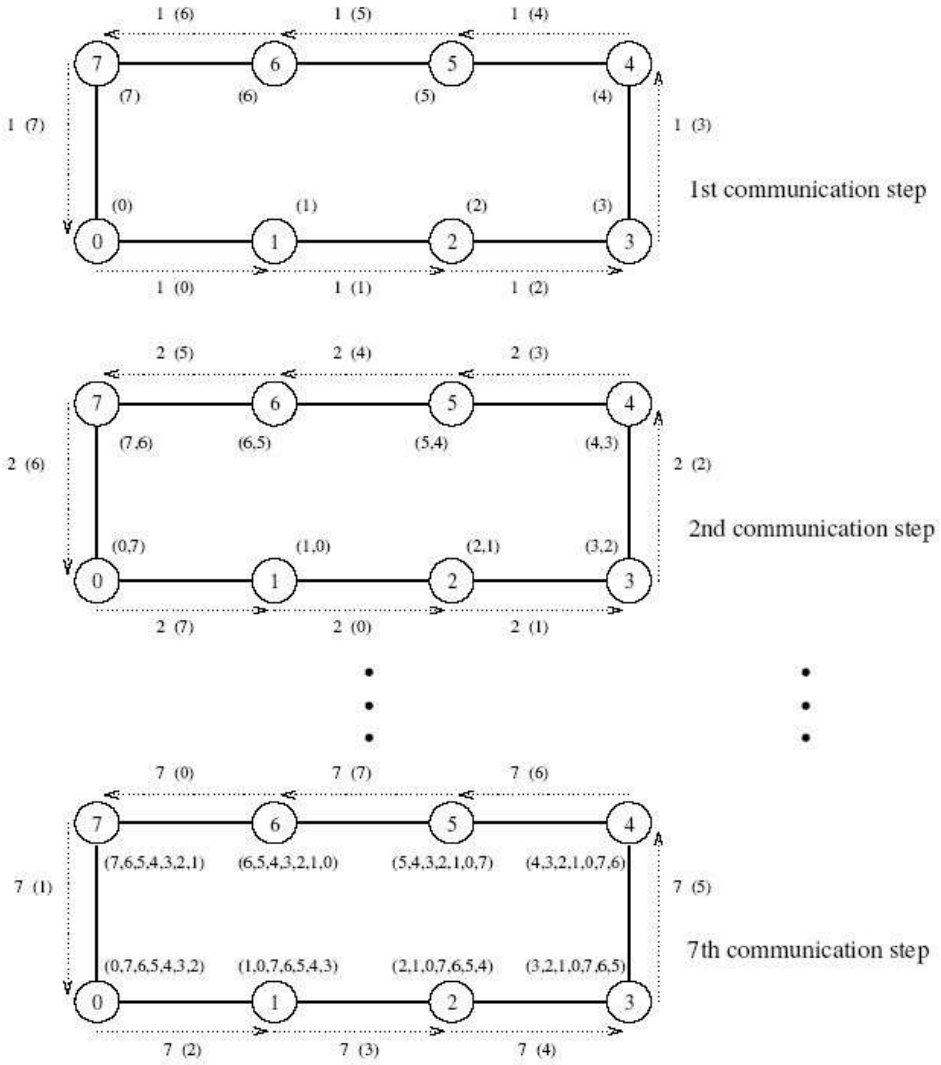


All-to-all broadcast and all-to-all reduction.

All-to-All Broadcast and Reduction on a Ring

- Simplest approach: perform *p* one-to-all broadcasts. This is not the most efficient way, though.
- Each node first sends to one of its neighbors the data it needs to broadcast.
- In subsequent steps, it forwards the data received from one of its neighbors to its other neighbor.
- The algorithm terminates in *p-1* steps.

All-to-All Broadcast and Reduction on a Ring

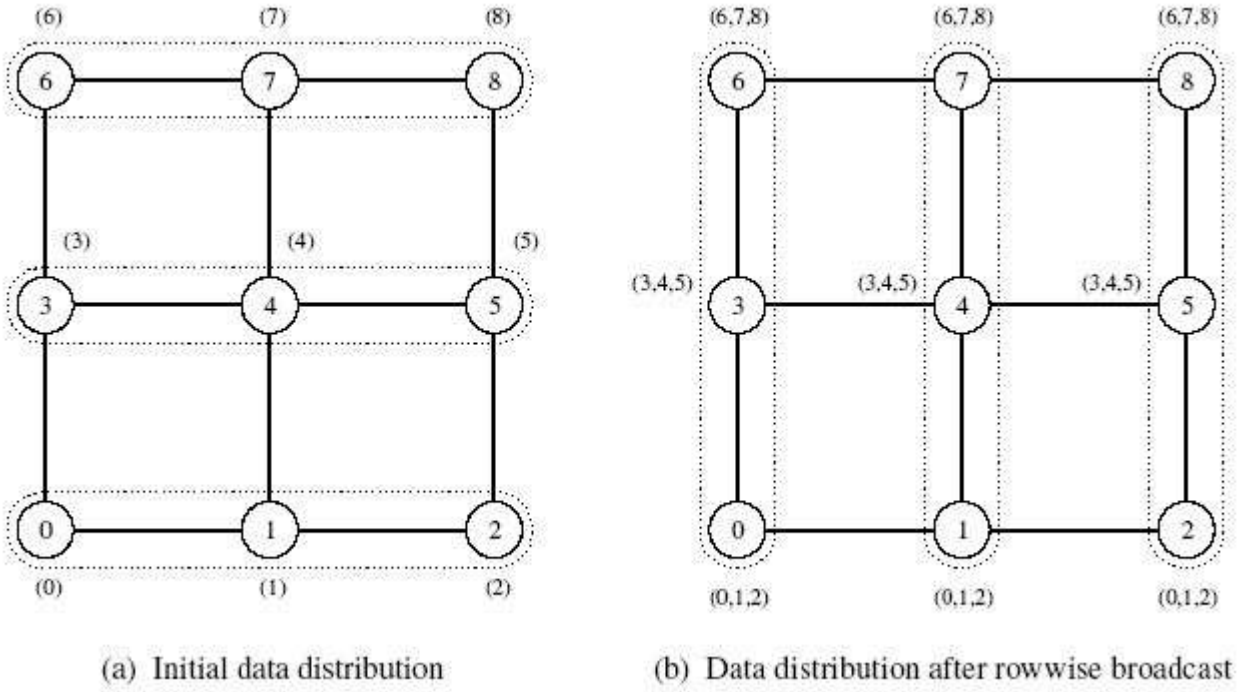


All-to-all broadcast on an eight-node ring.

All-to-all Broadcast on a Mesh

- Performed in two phases - in the first phase, each row of the mesh performs an all-to-all broadcast using the procedure for the linear array.
- In this phase, all nodes collect $\sqrt{p}$ messages corresponding to the $\sqrt{p}$ nodes of their respective rows. Each node consolidates this information into a single message of size $m\sqrt{p}$.
- The second communication phase is a columnwise all-to-all broadcast of the consolidated messages.

All-to-all Broadcast on a Mesh



(a) Initial data distribution

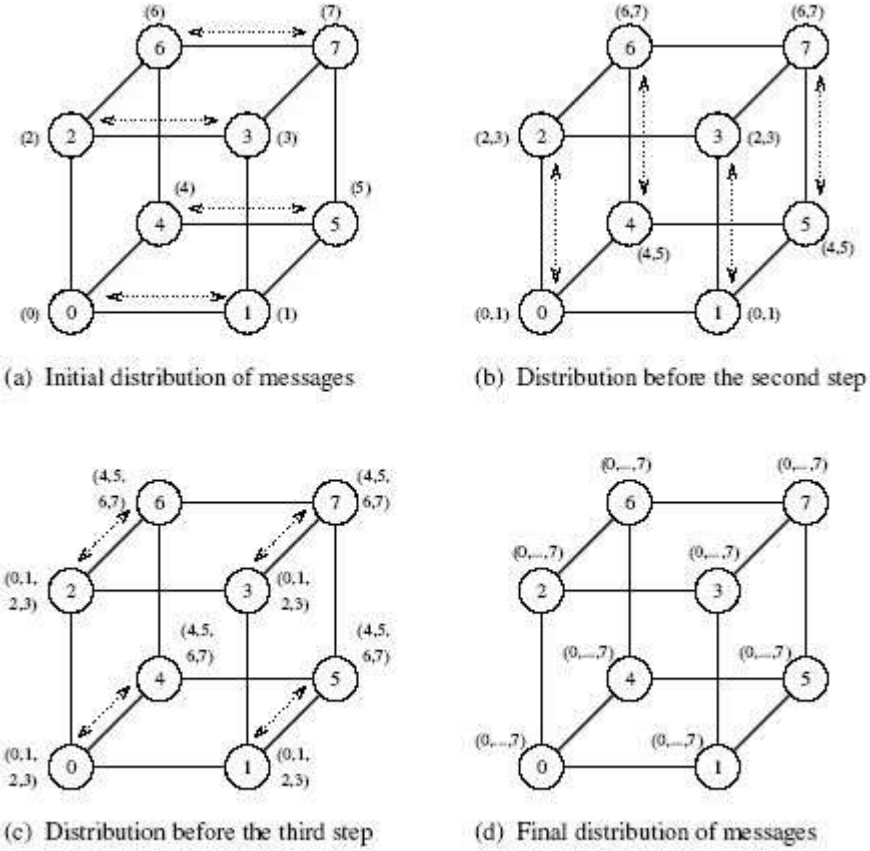
(b) Data distribution after rowwise broadcast

All-to-all broadcast on a *3 x 3* mesh. The groups of nodes communicating with each other in each phase are enclosed by dotted boundaries. By the end of the second phase, all nodes get (0,1,2,3,4,5,6,7) (that is, a message from each node).

All-to-all broadcast on a Hypercube

- Generalization of the mesh algorithm to *log p* dimensions.
- Message size doubles at each of the *log p* steps.

All-to-all broadcast on a Hypercube



All-to-all broadcast on an eight-node hypercube.

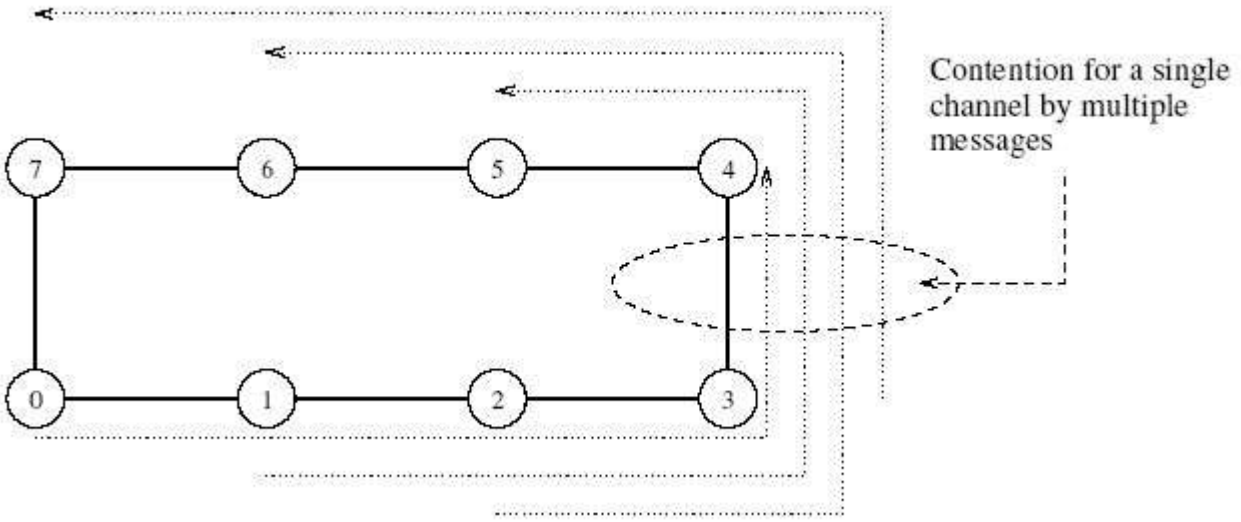
All-to-all Reduction

- Similar communication pattern to all-to-all broadcast, except in the reverse order.
- On receiving a message, a node must combine it with the local copy of the message that has the same destination as the received message before forwarding the combined message to the next neighbor.

All-to-all broadcast: Notes

- All of the algorithms presented above are asymptotically optimal in message size.
- It is not possible to port algorithms for higher dimensional networks (such as a hypercube) into a ring because this would cause contention.

All-to-all broadcast: Notes



Contention for a channel when the hypercube is mapped onto a ring.

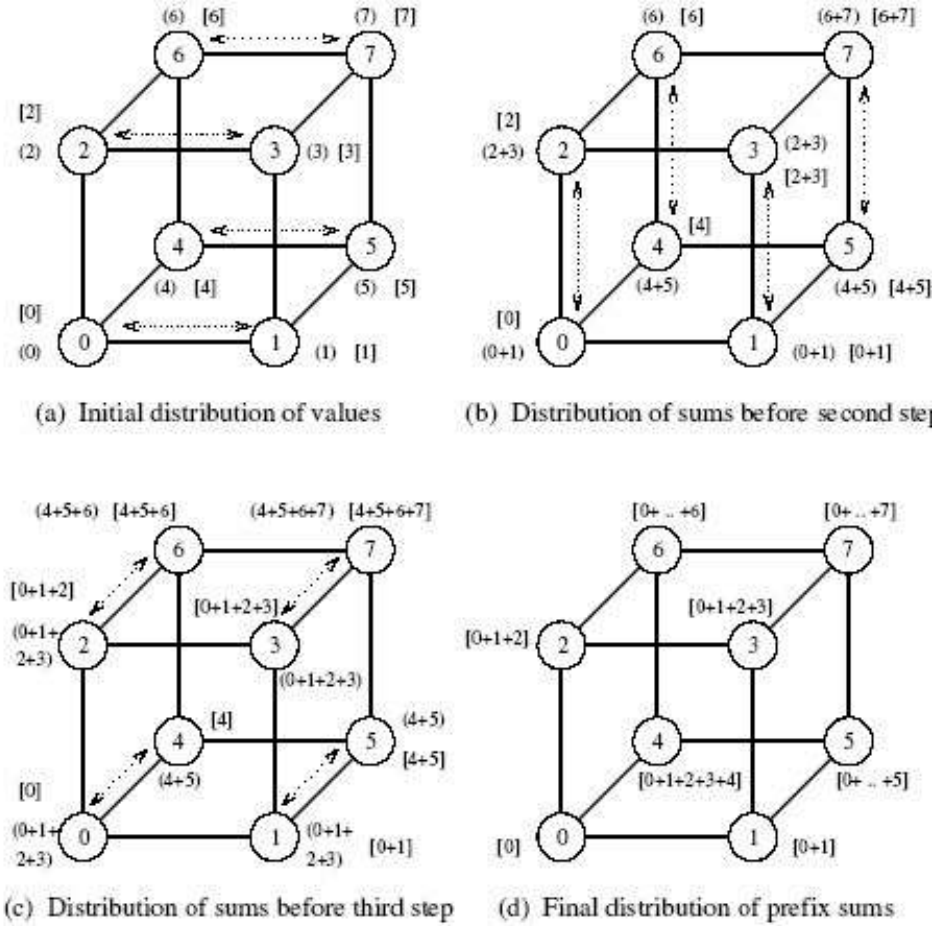
All-Reduce and Prefix-Sum Operations

- In all-reduce, each node starts with a buffer of size *m* and the final results of the operation are identical buffers of size *m* on each node that are formed by combining the original *p* buffers using an associative operator.
- Identical to all-to-one reduction followed by a one-to-all broadcast. This formulation is not the most efficient. Uses the pattern of all-to-all broadcast, instead. The only difference is that message size does not increase here. Time for this operation is $(t_s + t_w m) \log p$.
- Different from all-to-all reduction, in which *p* simultaneous all-to-one reductions take place, each with a different destination for the result.

The Prefix-Sum Operation

- Given p numbers $n_0, n_1, \ldots, n_{p-1}$ (one on each node), the problem is to compute the sums $s_k = \sum_{i=0}^{k} n_i$ for all k between 0 and $p-1$.
- Initially, n_k resides on the node labeled k, and at the end of the procedure, the same node holds S_k.

The Prefix-Sum Operation



(a) Initial distribution of values

(b) Distribution of sums before second step

(c) Distribution of sums before third step

(d) Final distribution of prefix sums

Computing prefix sums on an eight-node hypercube. At each node, square brackets show the local prefix sum accumulated in the result buffer and parentheses enclose the contents of the outgoing message buffer for the next step.

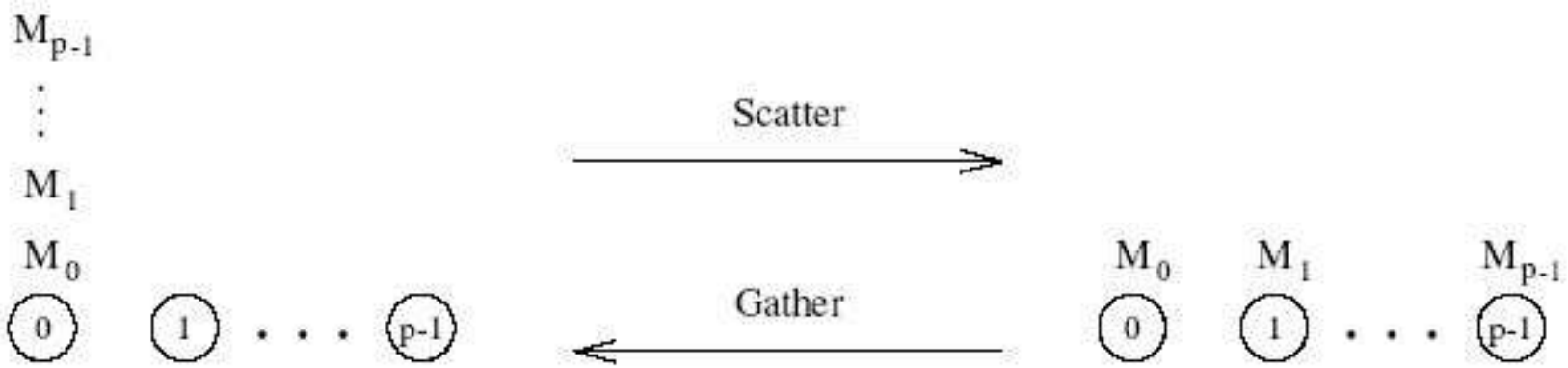
The Prefix-Sum Operation

- The operation can be implemented using the all-to-all broadcast kernel.
- We must account for the fact that in prefix sums the node with label k uses information from only the k-node subset whose labels are less than or equal to k.
- This is implemented using an additional result buffer. The content of an incoming message is added to the result buffer only if the message comes from a node with a smaller label than the recipient node.
- The contents of the outgoing message (denoted by parentheses in the figure) are updated with every incoming message.

Scatter and Gather

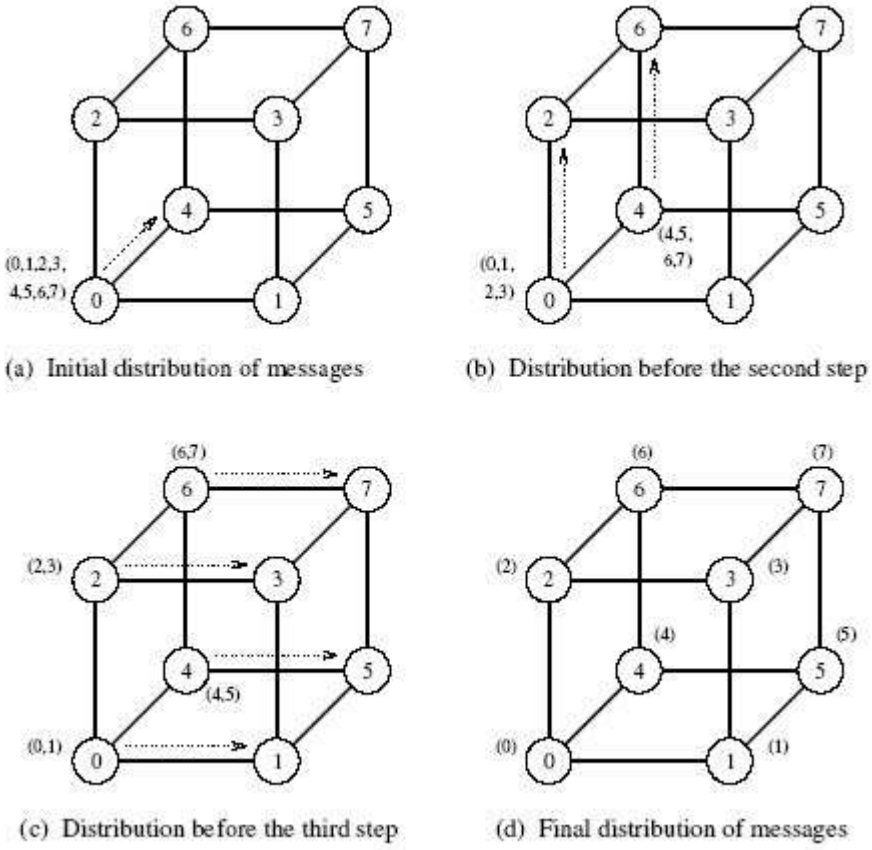
- In the *scatter* operation, a single node sends a unique message of size *m* to every other node (also called a one-to-all personalized communication).
- In the *gather* operation, a single node collects a unique message from each node.
- While the scatter operation is fundamentally different from broadcast, the algorithmic structure is similar, except for differences in message sizes (messages get smaller in scatter and stay constant in broadcast).
- The gather operation is exactly the inverse of the scatter operation and can be executed as such.

Gather and Scatter Operations



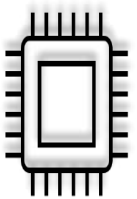
Scatter and gather operations.

Example of the Scatter Operation



The scatter operation on an eight-node hypercube.

References Used



- Ananth Grama, Anshul Gupta, George Karypis, Vipin Kumar , —Introduction to Parallel Computing,Pearson Education, Second Edition, 2007.