Basic Communication Operations

Alexandre David
B2-206

Today

- One-to-all broadcast & all-to-one reduction (4.1).
- All-to-all broadcast and reduction (4.2).
- All-reduce and prefix-sum operations (4.3).

Collective Communication Operations

- Represent regular communication patterns.
- Used extensively in most data-parallel algorithms.
- Critical for efficiency.
- Available in most parallel libraries.
- Very useful to “get started” in parallel processing.

Collective: involve group of processors.

The efficiency of data-parallel algorithms depends on the efficient implementation of these operations.

Recall: t_s+mt_w time for exchanging a *m*-word message with cut-through routing.

All processes participate in a single **global** interaction operation or subsets of processes in **local** interactions.

Goal of this chapter: good algorithms to implement commonly used communication patterns.

Reminder

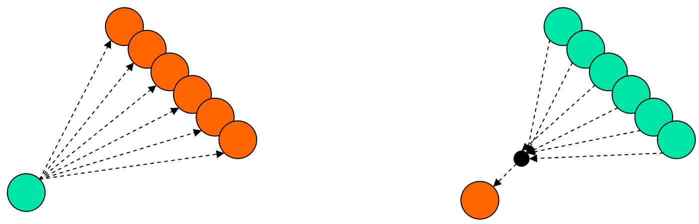
- Result from previous analysis:
  - Data transfer time is roughly the same between *all pairs* of nodes.
  - Homogeneity true on modern hardware (randomized routing, cut-through routing...).
    - t_s+mt_w
    - Adjust t_w for congestion: effective t_w.
- Model: bidirectional links, single port.
- Communication with point-to-point primitives.

Broadcast/Reduction

- One-to-all broadcast:
  - Single process sends identical data to all (or subset of) processes.
- All-to-one reduction:
  - Dual operation.
  - *P* processes have *m* words to send to one destination.
  - Parts of the message need to be *combined*.

Reduction can be used to find the sum, product, maximum, or minimum of sets of nmbers.

Broadcast/Reduction



Broadcast   Reduce

28-02-2006   Alexandre David, MVP'06   6

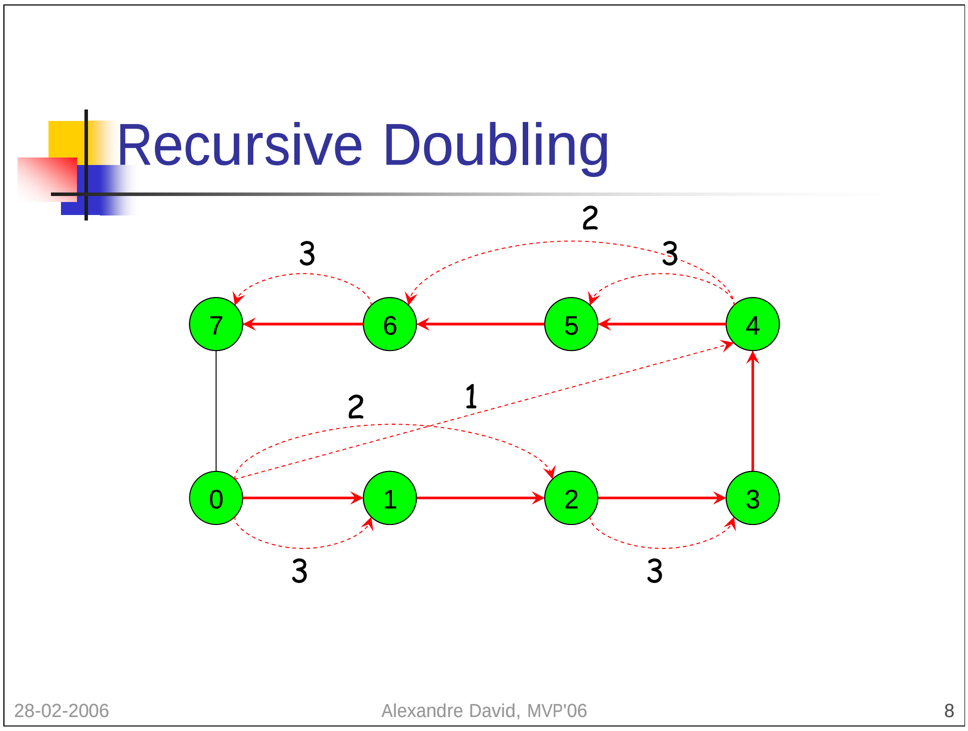
This is the logical view, what happens from the programmer's perspective.

One-to-All Broadcast – Ring/Linear Array

- Naïve approach: send sequentially.
  - Bottleneck.
  - Poor utilization of the network.
- Recursive doubling:
  - Broadcast in log*p* steps (instead of p).
  - Divide-and-conquer type of algorithm.
  - Reduction is similar.

Source *process* is the bottleneck. Poor utilization: Only connections between single pairs of nodes are used at a time.

Recursive doubling: All processes that have the data can send it again.



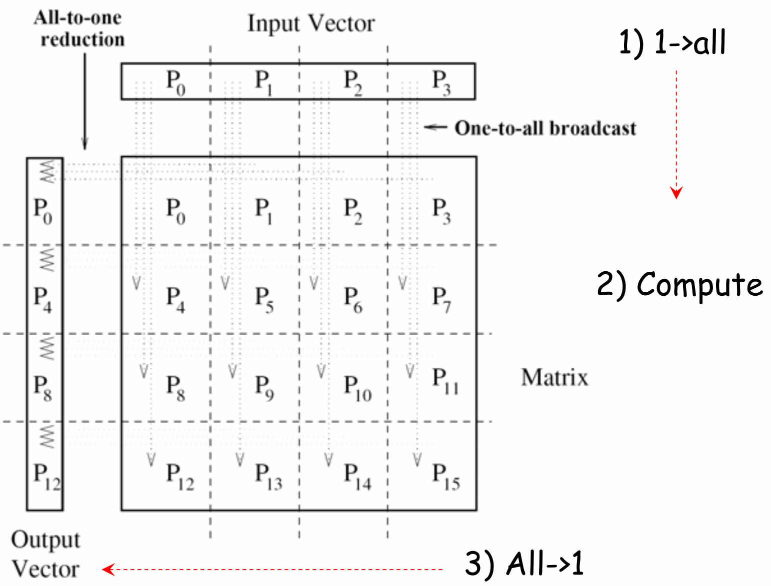
Note:

•The nodes do not snoop the messages going “through” them. Messages are forwarded but the processes are not notified of this because they are not destined to them.

•Choose carefully destinations: furthest.

•Reduction symmetric: Accumulate results and send with the same pattern.

Example: Matrix*Vector

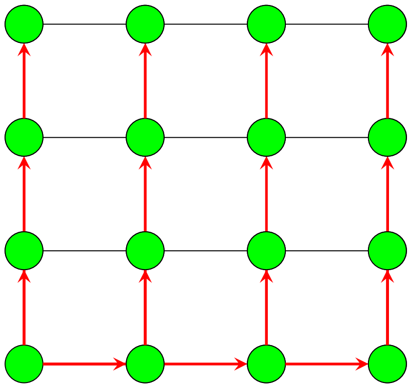


Although we have a matrix & a vector the broadcast are done on arrays.

One-to-All Broadcast – Mesh

- Extensions of the linear array algorithm.
  - Rows & columns = arrays.
  - Broadcast on a row, broadcast on columns.
  - Similar for reductions.
  - Generalize for higher dimensions (cubes…).

Broadcast on a Mesh



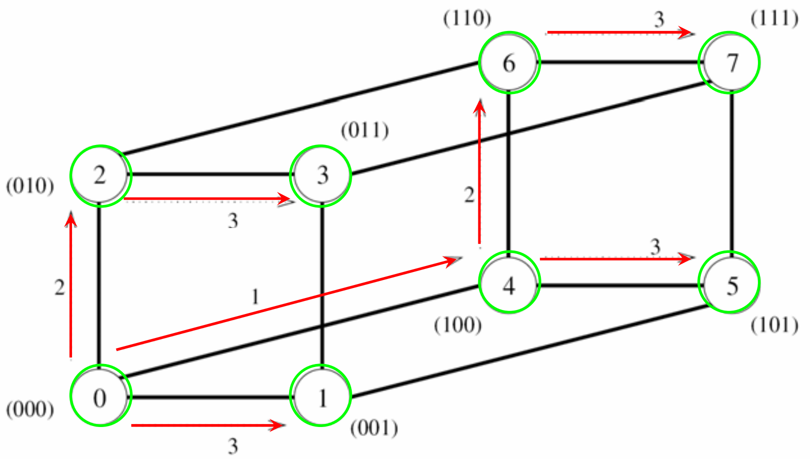
28-02-2006 Alexandre David, MVP'06 11

1. Broadcast like linear array.
2. Every node on the linear array has the data and broadcast on the columns with the linear array algorithm, *in parallel*.

One-to-All Broadcast – Hypercube

- Hypercube with 2^d nodes = d-dimensional mesh with 2 nodes in each direction.
- Similar algorithm in d steps.
- Also in log*p* steps.
- Reduction follows the same pattern.

Broadcast on a Hypercube



Better for congestion: Use different links every time. Forwarding in parallel again.

All-to-One Broadcast – Balanced Binary Tree

- Processing nodes = leaves.
- Hypercube algorithm maps well.
- Similarly good w.r.t. congestion.

Broadcast on a Balanced Binary Tree

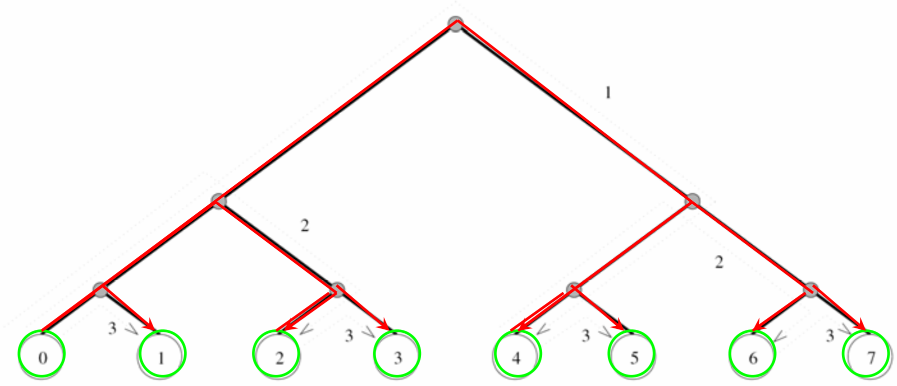


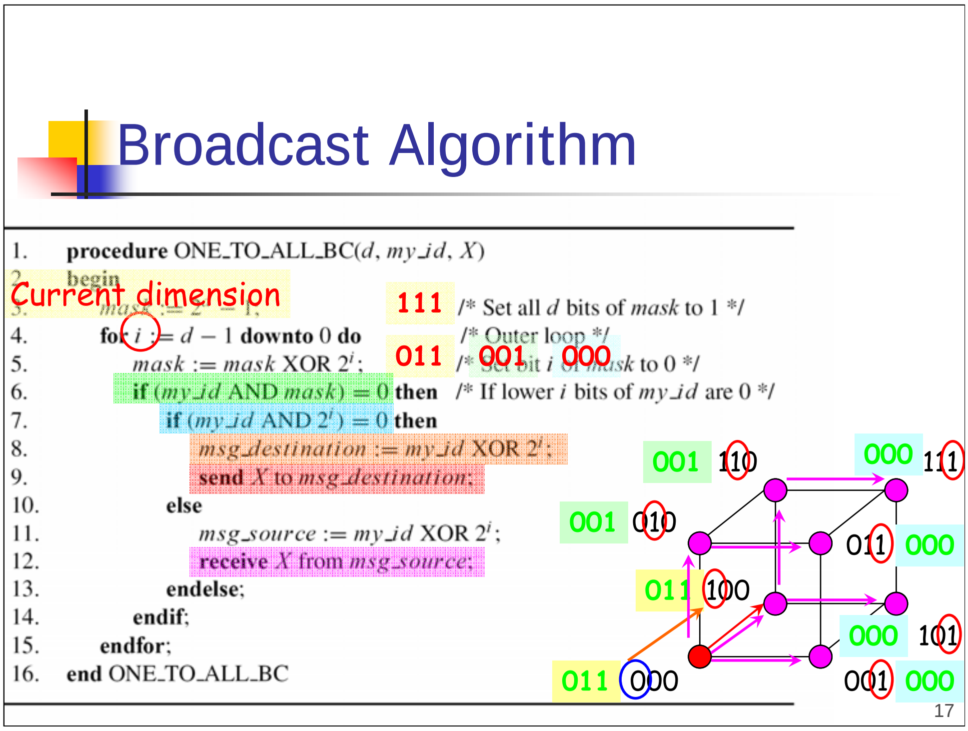
Figure 4.7 One-to-all broadcast on an eight-node tree.

Divide-and-conquer type of algorithm again.

Algorithms

- So far we saw pictures.
- Not enough to implement.
- Precise description
  - to implement.
  - to analyze.
- Description for hypercube.
- Execute the following procedure on all the nodes.

For sake of simplicity, the number of nodes is a power of 2.

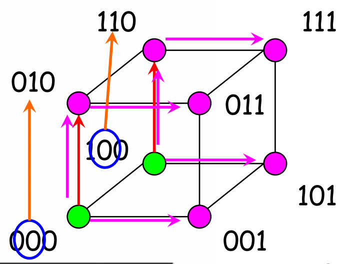


my_id is the label of the node the procedure is executed on. The procedure performs **d** communication steps, one along each dimension of the hypercube.

Nodes with zero in **i** least significant bits (of their labels) participate in the communication.

Broadcast Algorithm

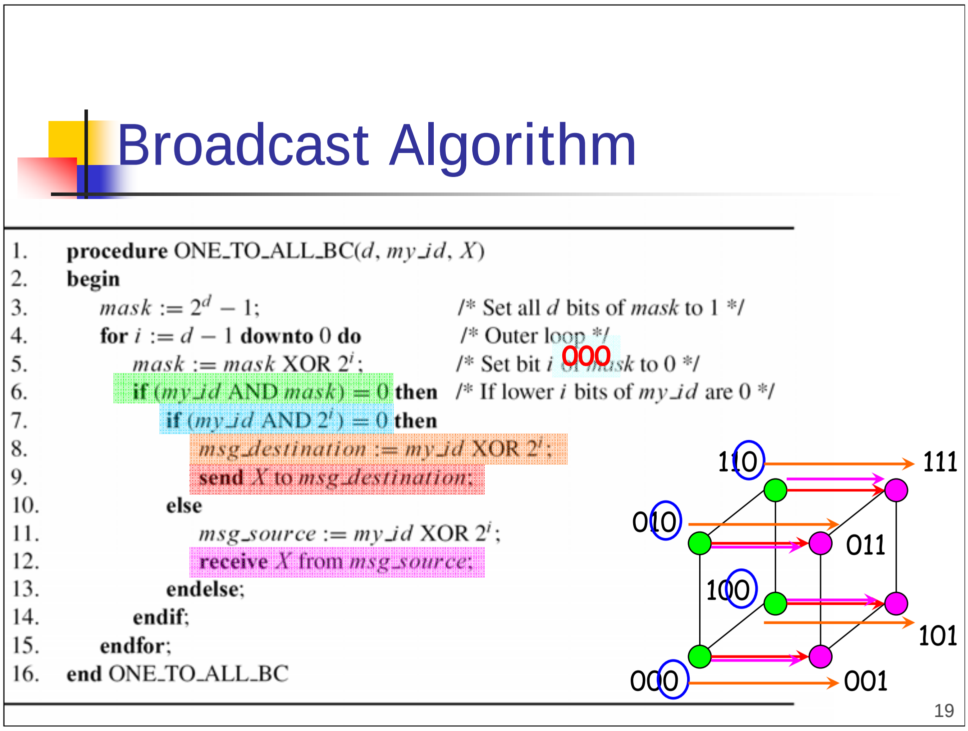
```
procedure ONE_TO_ALL_BC(d, my_id, X)
begin
    mask := 2^d - 1;                          /* Set all d bits of mask to 1 */
    for i := d - 1 downto 0 do                /* Outer loop */
        mask := mask XOR 2^i;                 /* Set bit i of mask to 0 */
        if (my_id AND mask) = 0 then          /* If lower i bits of my_id are 0 */
            if (my_id AND 2^i) = 0 then
                msg_destination := my_id XOR 2^i;
                send X to msg_destination;
            else
                msg_source := my_id XOR 2^i;
                receive X from msg_source;
            endelse;
        endif;
    endfor;
end ONE_TO_ALL_BC
```



18

my_id is the label of the node the procedure is executed on. The procedure performs **d** communication steps, one along each dimension of the hypercube.

Nodes with zero in **i** least significant bits (of their labels) participate in the communication.



my_id is the label of the node the procedure is executed on. The procedure performs **d** communication steps, one along each dimension of the hypercube.

Nodes with zero in **i** least significant bits (of their labels) participate in the communication.

Notes:

•Every node has to know when to communicate, i.e., call the procedure.

•The procedure is distributed and requires only point-to-point synchronization.

•Only from node 0.

Algorithm For Any Source

```
procedure GENERAL_ONE_TO_ALL_BC(d, my_id, source, X)
begin
   my_virtual_id := my_id XOR source;
   mask := 2^d - 1;
   for i := d - 1 downto 0 do      /* Outer loop */
      mask := mask XOR 2^i;    /* Set bit i of mask to 0 */
      if (my_virtual_id AND mask) = 0 then
         if (my_virtual_id AND 2^i) = 0 then
            virtual_dest := my_virtual_id XOR 2^i;
            send X to (virtual_dest XOR source);
   /* Convert virtual_dest to the label of the physical destination */
         else
            virtual_source := my_virtual_id XOR 2^i;
            receive X from (virtual_source XOR source);
   /* Convert virtual_source to the label of the physical source */
         endelse;
   endfor;
end GENERAL_ONE_TO_ALL_BC
```

20

XOR the source = renaming relative to the source. Still works because of the sub-cube property: changing 1 bit = navigate on one dimension, keep a set of equal bits = sub-cube.

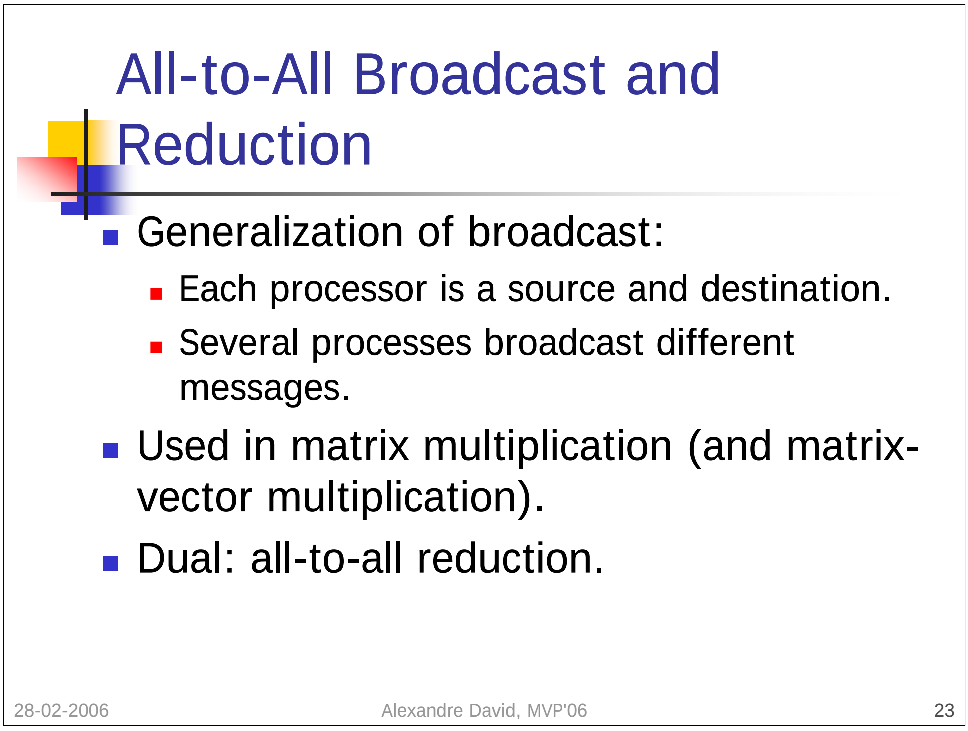
Reduce Algorithm

```
procedure ALL_TO_ONE_REDUCE(d, my_id, m, X, sum)
begin
    for j := 0 to m − 1 do sum[j] := X[j];
    mask := 0;
    for i := 0 to d − 1 do
        /* Select nodes whose lower i bits are 0 */
        if (my_id AND mask) = 0 then
            if (my_id AND 2^i) ≠ 0 then
                msg_destination := my_id XOR 2^i;
                receive X from msg_source;
                for j := 0 to m − 1 do
                    sum[j] :=sum[j] + X[j];
            endelse;
        mask := mask XOR 2^i;    /* Set bit i of mask to 1 */
    endfor;
end ALL_TO_ONE_REDUCE
```

In a nutshell:
reverse the previous one.

Cost Analysis

p processes → log$p$ steps (point-to-point transfers in parallel).
Each transfer has a time cost of $t_s+t_w m$.
Total time: $T=(t_s+t_w m)\log p$.



How to do it?

If performed naively, it may take up to p times as long as a one-to-all broadcast (for p processors).

Possible to concatenate all messages that are going through the same path (reduce time because fewer t_s).

All-to-All Broadcast and Reduction

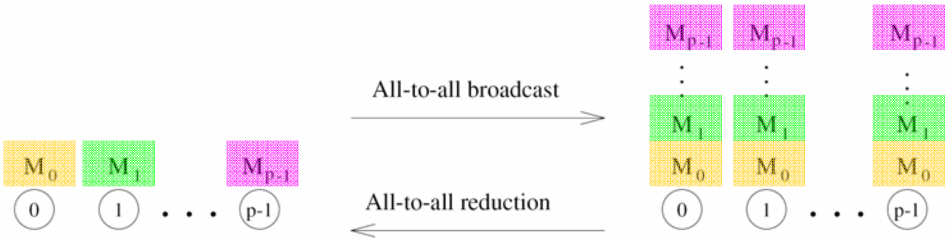
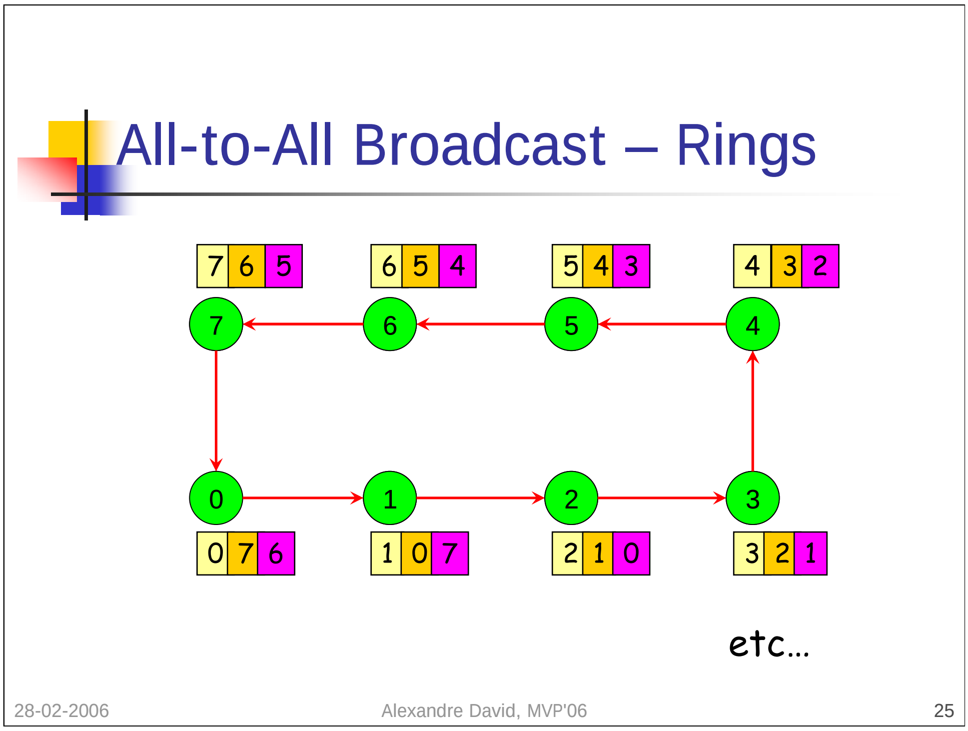


Figure 4.8 All-to-all broadcast and all-to-all reduction.



All communication links can be kept busy until the operation is complete because each node has some information to pass. One-to-all in log*p* steps, all-to-all in *p-1* steps instead of *p* log*p* (naïve).

How to do it for linear arrays? If we have bidirectional links (assumption from the beginning), we can use the same procedure.

All-to-All Broadcast Algorithm

```
procedure ALL_TO_ALL_BC_RING(my_id, my_msg, p, result)
begin
    left := (my_id − 1) mod p;
    right := (my_id + 1) mod p;
    result := my_msg;
    msg := result;
    for i := 1 to p − 1 do
        send msg to right;
        receive msg from left;
        result := result ∪ msg;
    endfor;
end ALL_TO_ALL_BC_RING
```

Ring: mod *p*.
Receive & send – point-to-point.
Initialize the loop.
Forward msg.
Accumulate result.

Algorithm 4.4 All-to-all broadcast on a p-node ring.

All-to-All Reduce Algorithm

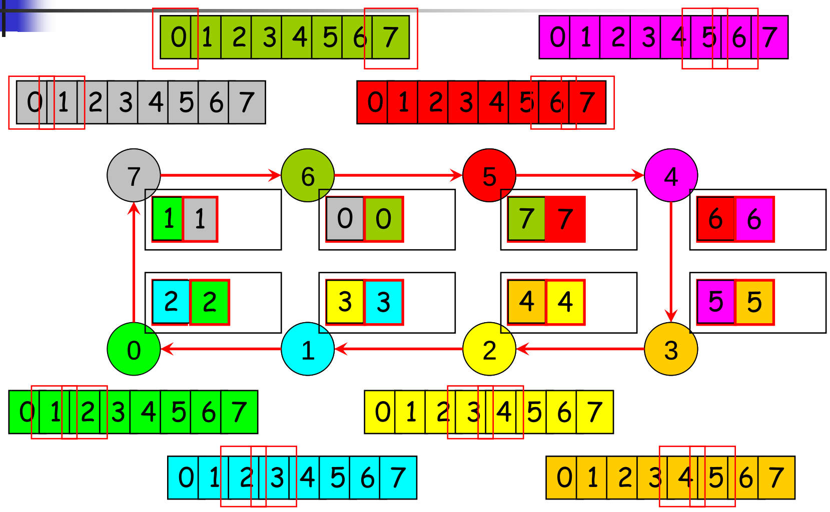
```
procedure ALL_TO_ALL_RED_RING(my_id, my_msg, p, result)
begin
    left := (my_id − 1) mod p;
    right := (my_id + 1) mod p;
    recv := 0;
    for i := 1 to p − 1 do
        j := (my_id + i) mod p;
        temp := msg[j] + recv;
        send temp to left;
        receive recv from right;
    endfor;
    result := msg[my_id] + recv;
end ALL_TO_ALL_RED_RING
```

Accumulate and forward.

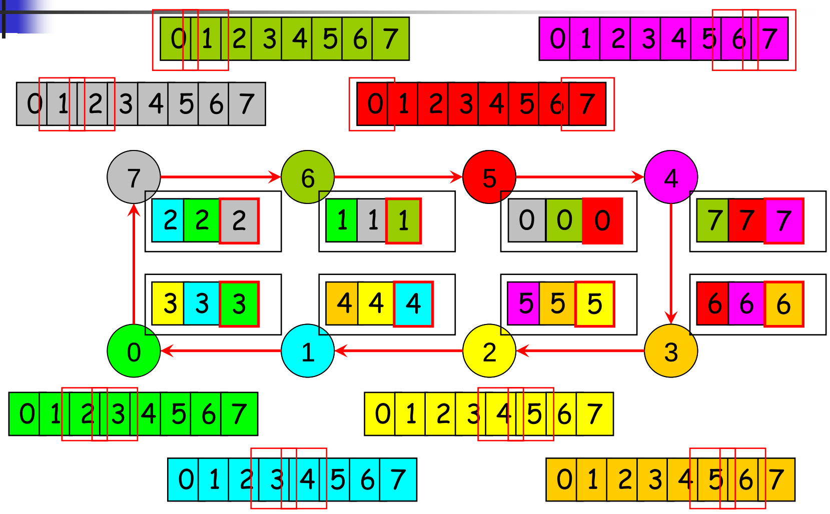
Last message for *my_id*.

Algorithm 4.5 All-to-all reduction on a p-node ring.

All-to-All Reduce – Rings



All-to-All Reduce – Rings



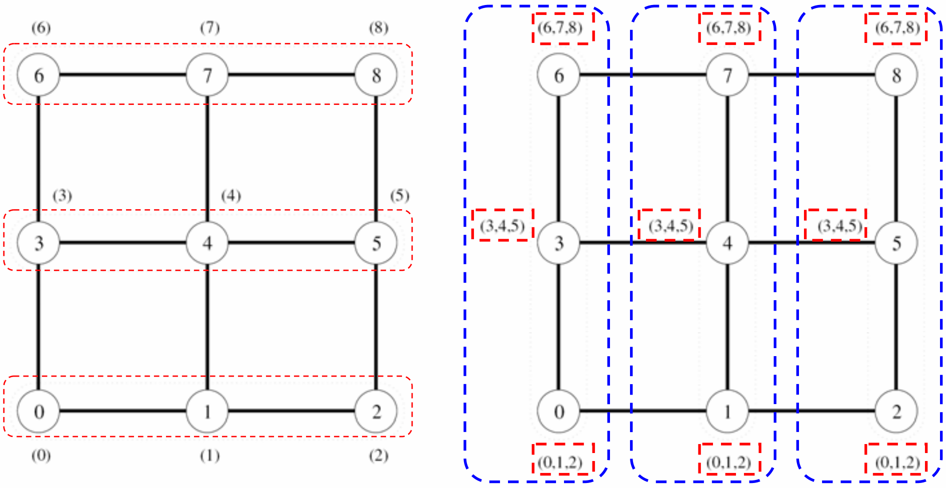
28-02-2006 Alexandre David, MVP'06 29

p-1 steps.

All-to-All Broadcast – Meshes

- Two phases:
  - All-to-all on rows – messages size m.
    - Collect sqrt(p) messages.
  - All-to-all on columns – messages size sqrt(p)*m.

All-to-All Broadcast – Meshes



Algorithm

```
procedure ALL_TO_ALL_BC_MESH(my_id, my_msg, p, result)
begin
/* Communication along rows */
    left := my_id − (my_id mod √p) + (my_id − 1)mod √p;
    right := my_id − (my_id mod √p) + (my_id + 1) mod √p;
    result := my_msg;
    msg := result;
    for i := 1 to √p − 1 do
        send msg to right;
        receive msg from left;
        result := result ∪ msg;
    endfor;
/* Communication along columns */
    up := (my_id − √p) mod p;
    down := (my_id + √p) mod p;
    msg := result;
    for i := 1 to √p − 1 do
        send msg to down;
        receive msg from up;
        result := result ∪ msg;
    endfor;
end ALL_TO_ALL_BC_MESH
```

All-to-All Broadcast - Hypercubes

- Generalization of the mesh algorithm to log*p* dimensions.
- Message size doubles at every step.
- Number of steps: log*p*.

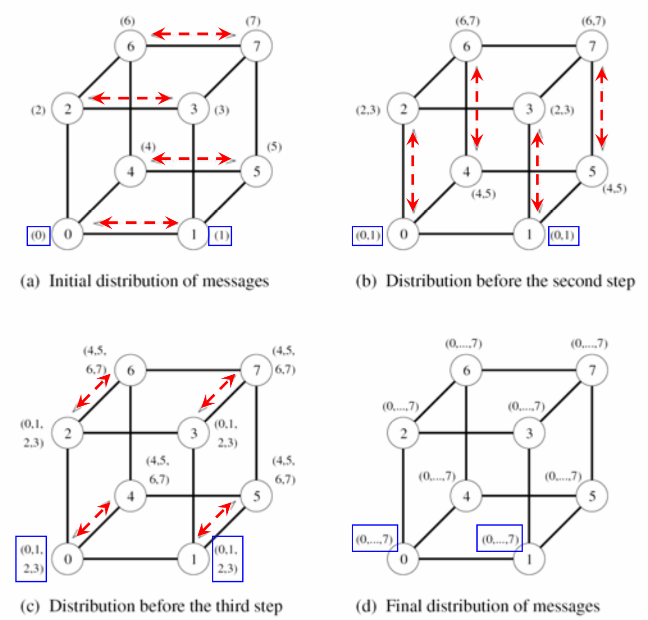
Remember the 2 extremes:

•Linear array: p nodes per (1) dimension – p^1.

•Hypercubes: 2 nodes per log*p* dimensions – $2^{\log p}$.

And in between 2-D mesh sqrt(p) nodes per (2) dimensions – $\text{sqrt}(p)^2$.

All-to-All Broadcast – Hypercubes



Algorithm

```
procedure ALL_TO_ALL_BC_HCUBE(my_id, my_msg, d, result)
begin
    result := my_msg;
    for i := 0 to d − 1 do
        partner := my_id XOR 2^i;
        send result to partner;
        receive msg from partner;
        result := result ∪ msg;
    endfor;
end ALL_TO_ALL_BC_HCUBE
```

Loop on the dimensions

Exchange messages

Forward (double size)

Algorithm 4.7 All-to-all broadcast on a d-dimensional hypercube.

At every step we have a broadcast on sub-cubes. The size of the sub-cubes doubles at every step and all the nodes exchange their messages.

All-to-All Reduction – Hypercubes

```
procedure ALL_TO_ALL_RED_HCUBE(my_id, msg, d, result)
begin
    recloc := 0;
    for i := d − 1 to 0 do
        partner := my_id XOR 2^i;
        j := my_id AND 2^i;
        k := (my_id XOR 2^i) AND 2^i;
        senloc := recloc + k;
        recloc := recloc + j;
        send msg[senloc .. senloc + 2^i − 1] to partner;
        receive temp[0 .. 2^i − 1] from partner;
        for j := 0 to 2^i − 1 do
            msg[recloc + j] := msg[recloc + j] + temp[j];
        endfor;
    endfor;
    result := msg[my_id];
end ALL_TO_ALL_RED_HCUBE
```

Similar pattern in reverse order.

Combine results

Algorithm 4.8 All-to-all broadcast on a d-dimensional hypercube. AND and XOR are bitwise logical-and and exclusive-or operations, respectively.

Cost Analysis (Time)

- Ring:
  - $T=(t_s + t_w m)(p-1)$.
- Mesh:
  - $T=(t_s + t_w m)(\sqrt{p}-1)+(t_s + t_w m\sqrt{p})(\sqrt{p}-1)$
    $= 2t_s(\sqrt{p} - 1) + t_w m(p-1)$.
- Hypercube:
  - $$T = \sum_{i=1}^{\log p}(t_s + 2^{i-1}t_w m)$$
    $$= t_s \log p + t_w m(p-1).$$

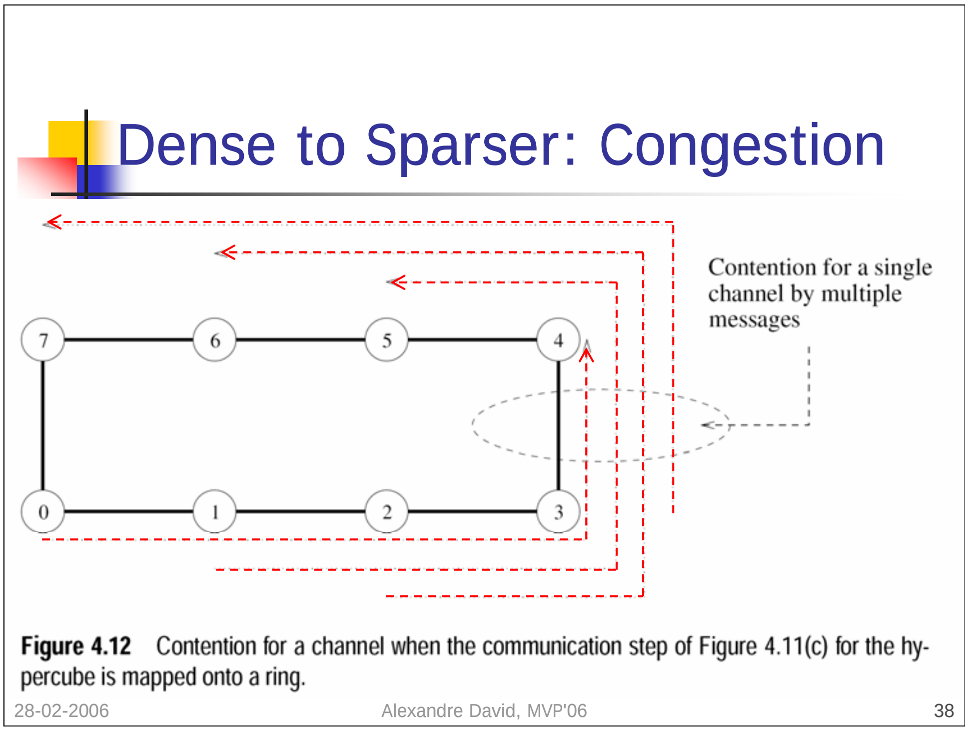
log$p$ steps
message of size $2^{i-1}m$.

28-02-2006 Alexandre David, MVP'06 37

Lower bound for the communication time of all-to-all broadcast for parallel computers on which a node can communicate on only one of its ports at a time = $t_w m(p-1)$. Each node receives at least $m(p-1)$ words of data. That's for **any** architecture.

The straight-forward algorithm for the simple ring architecture is interesting: It is a sequence of p one-to-all broadcasts with different sources every time. The broadcasts are pipelined. That's common in parallel algorithms.

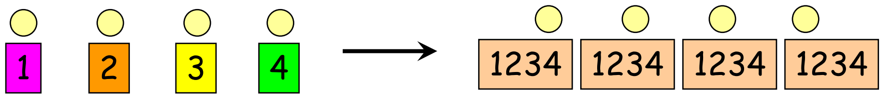
We cannot use the hypercube algorithm on smaller dimension topologies because of congestion.



Contention because communication is done on links with single ports. Contention is in the sense of the access to the link. The result is congestion on the traffic.

All-Reduce

- Each node starts with a buffer of size *m*.
- The final result is the same combination of all buffers on every node.
- Same as all-to-one reduce + one-to-all broadcast.
- Different from all-to-all reduce.



All-to-all reduce combines *p* different messages on *p* different nodes. All-reduce combines 1 message on *p* different nodes.

All-Reduce Algorithm

- Use all-to-all broadcast but
  - Combine messages instead of concatenating them.
  - The size of the messages does not grow.
  - Cost (in log$p$ steps): $T=(t_s+t_w m)\log p$.

Prefix-Sum

- 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 is on the node labeled k, and at the end, the same node holds S_k.

28-02-2006 Alexandre David, MVP'06 41

This is a reminder.

Prefix-Sum Algorithm

```
procedure PREFIX_SUMS_HCUBE(my_id, my_number, d, result)
begin
    result := my_number;
    msg := result;
    for i := 0 to d − 1 do
        partner := my_id XOR 2^i;
        send msg to partner;
        receive number from partner;
        msg := msg + number;
        if (partner < my_id) then result := result + number;
    endfor;
end PREFIX_SUMS_HCUBE
```

All-reduce

Prefix-sum

Algorithm 4.9 Prefix sums on a d-dimensional hypercube.

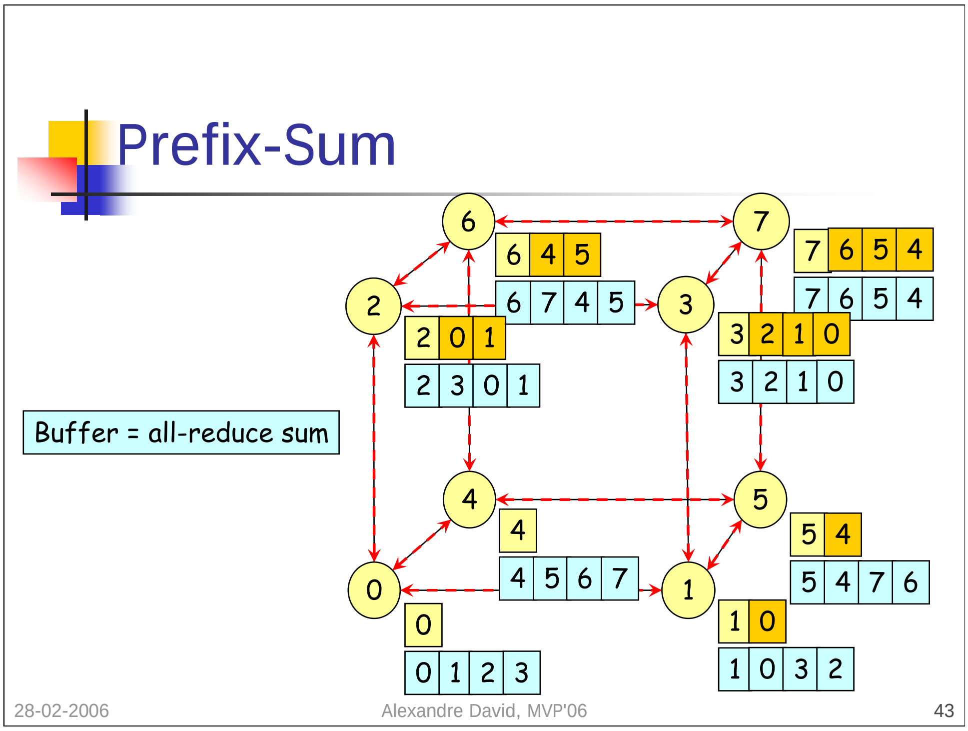


Figure in the book is messed up.