Basic Communication Patterns — building blocks

- ring, 2D-mesh, HC
- SF and CT routing schemes
- bidirectional links
- processor can send on only 1 link at a time
- processor can receive ——————"——————
- for small m, transfer time similar for SF & CT
- for $m \gg l$, distance between processors unimportant for CT

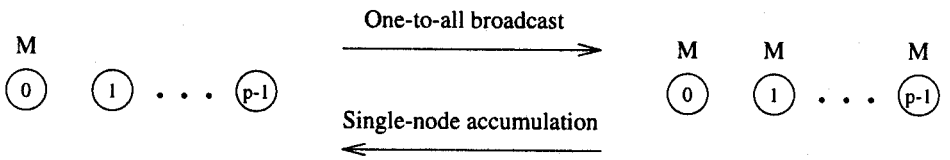


Figure 3.1 One-to-all broadcast and single-node accumulation. [duals]

- Fig 3-1:

  1 → all broadcast & single-node accumulation:

  - matrix vector multiplication
  - Gaussian elimination
  - shortest path
  - vector inner product
  - ------

- 1 → all broadcast
- circular shift
- all → all broadcast, reduction, prefix sums
- 1 → all personalized communication
- all → all personalized communication

$$\bullet\ T_{one \to all} = (t_s + t_w m)\left\lceil \frac{p}{2} \right\rceil$$

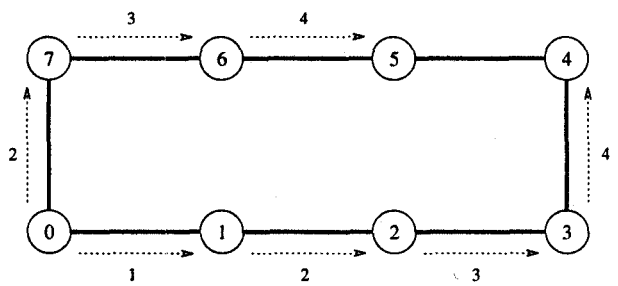


Figure 3.2 One-to-all broadcast on an eight-processor ring with SF routing. Processor 0 is the source of the broadcast. Each message transfer step is shown by a numbered, dotted arrow from the source of the message to its destination. The number on an arrow indicates the time step during which the message is transferred.

$n \times n$ matrix mult $n \times 1$ vector, on a mesh of processors

$$\begin{bmatrix} \quad \end{bmatrix} \times \begin{bmatrix} \quad \end{bmatrix}$$

- treat each column of the mesh as an n-processor ring

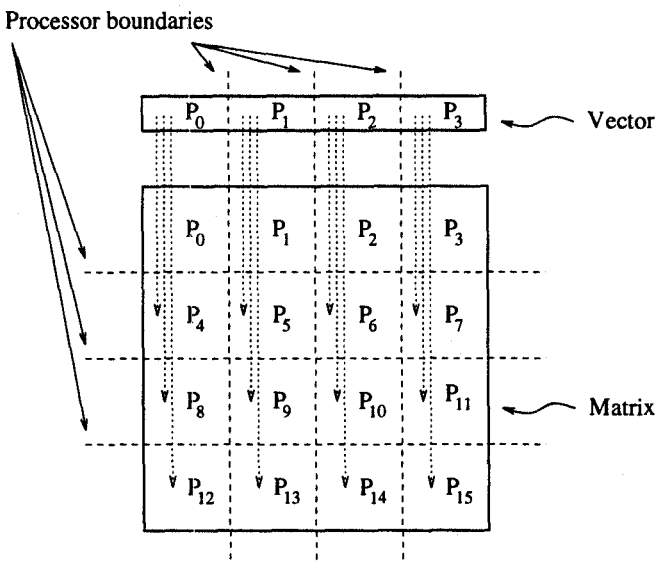


Figure 3.3 One-to-all broadcast in the multiplication of a 4×4 matrix with a 4×1 vector.

① 1→ all broadcast in row

② 1→ all broadcast in columns in parallel

$$T_{1\to all} = 2\left(t_s + t_w m\right)\left\lceil \frac{\sqrt{p}}{2} \right\rceil$$

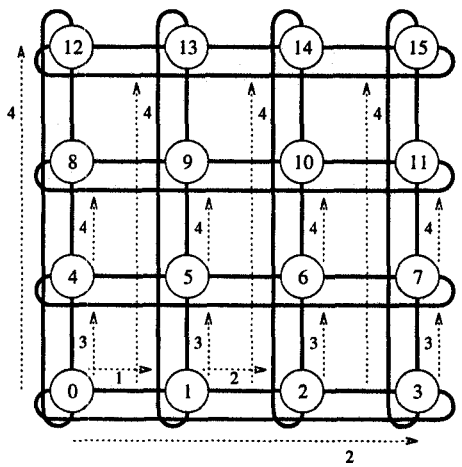


Figure 3.4 One-to-all broadcast on a 16-processor mesh with SF routing.

For a 3-D mesh,

$$T_{1\to all} = 3\left(t_s + t_w m\right)\left\lceil \frac{p^{1/3}}{2} \right\rceil$$

```
1.  procedure ONE_TO_ALL_BC(d, my_id, X)
2.  begin
3.      mask := 2^d − 1;              /* Set all d bits of mask to 1 */
4.      for i := d − 1 downto 0 do    /* Outer loop */
5.      begin
6.          mask := mask XOR 2^i;     /* Set bit i of mask to 0 */
7.          if (my_id AND mask) = 0 then
            /* If the lower i bits of my_id are 0 */
8.              if (my_id AND 2^i) = 0 then
9.              begin
10.                 msg_destination := my_id XOR 2^i;
11.                 send X to msg_destination;
12.             endif
13.             else
14.             begin
15.                 msg_source := my_id XOR 2^i;
16.                 receive X from msg_source;
17.             endelse;
18.     endfor;
19. end ONE_TO_ALL_BC
```

Program 3.1 One-to-all broadcast of a message X from processor 0 of a d-dimensional hypercube. AND and XOR are bitwise logical-and and exclusive-or operations, respectively.

phases on a
dimension
limensional
mesh with
ded for the
dimension

- communication in d steps, from highest to lowest dimension
- i: loop counter to track current dimension of communication
  processors with 0 in the i LSBs participate in comm along dim i
- mask: to determine which processors communicate in an iteration
  iteration 1: 011 (procs 0 & 4)
  iteration 2: 001 (procs 0, 2, 4, 6)
  iteration 3: 000 (all procs)

- '0' in bit posn i: ⟹ send
- '1' in bit posn i: ⟹ receive

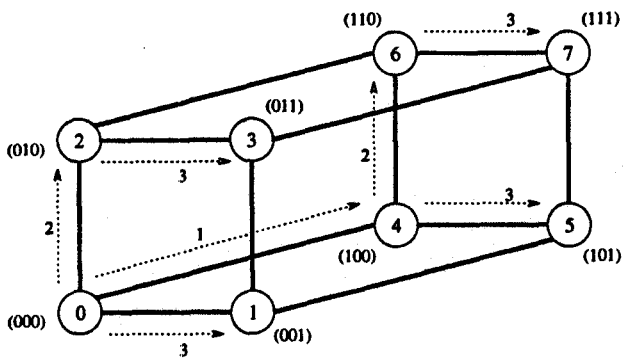


Figure 3.5 One-to-all broadcast on a three-dimensional hypercube. The binary representations of processor labels are shown in parentheses.

$$T_{1\to all} = (t_s + t_w m)\log p$$

```
1.  procedure GENERAL_ONE_TO_ALL_BC(d, my_id, source, X)
2.  begin
3.      my_virtual_id := my_id XOR source;
4.      mask := 2^d − 1;
5.      for i := d − 1 downto 0 do     /* Outer loop */
6.      begin
7.          mask := mask XOR 2^i;   /* Set bit i of mask to 0 */
8.          if (my_virtual_id AND mask) = 0 then
9.              if (my_virtual_id AND 2^i) = 0 then
10.             begin
11.                 virtual_dest := my_virtual_id XOR 2^i;
12.                 send X to (virtual_dest XOR source); /* Convert virtual_dest
                                        to the label of the physical destination */
13.             endif
14.             else
15.             begin
16.                 virtual_source := my_virtual_id XOR 2^i;
17.                 receive X from (virtual_source XOR source);
    /* Convert virtual_source to the label of the physical source */
18.             endelse;
19.     endfor;
20. end GENERAL_ONE_TO_ALL_BC
```

Program 3.2 One-to-all broadcast of a message X initiated by *source* in a d-dimensional hypercube. The AND and XOR operations are bitwise logical operations.

- dual (1→all BC)
- reverse order & direction of msgs.
- comm. from lowest to highest dimension

```
1.  procedure SINGLE_NODE_ACC(d, my_id, m, X, sum)
2.  begin
3.      for j := 0 to m − 1 do sum[j] := X[j];
4.      mask := 0;
5.      for i := 0 to d − 1 do
6.      begin /* Select processors whose lower i bits are 0 */
7.          if (my_id AND mask) = 0 then
8.              if (my_id AND 2^i) ≠ 0 then
9.              begin
10.                 msg_destination := my_id XOR 2^i;
11.                 send sum to msg_destination;
12.             endif
13.             else
14.             begin
15.                 msg_source := my_id XOR 2^i;
16.                 receive X from msg_source;
17.                 for j := 0 to m − 1 do
18.                     sum[j] := sum[j] + X[j];
19.             endelse;
20.         mask := mask XOR 2^i;   /* Set bit i of mask to 1 */
21.     endfor;
22. end SINGLE_NODE_ACC
```

Program 3.3 Single-node accumulation on a d-dimensional hypercube. Each processor contributes a msg X containing m words, & processor $\emptyset$ is the dest of the sum. AND & XOR are bitwise logical operations

- If msg is not routed in parts & comm. is allowed on only 1 link of each processor at a time, $T^{min}_{1\to all} = (t_s + t_w m)\log p = T^{HC}$ on any architecture
  - each proc. possessing data sends it to one that needs it, at each step
  - each msg only between directly connected procs
    ⇒ each step has min. duration
- $T^{min}_{1\to all}$ on HC with SF cannot be improved with CT routing due to exclusively nearest neighbor communication
- $T_{1\to all}$ on ring & mesh can be improved by using CT, instead of SF

CUT-THROUGH ROUTING

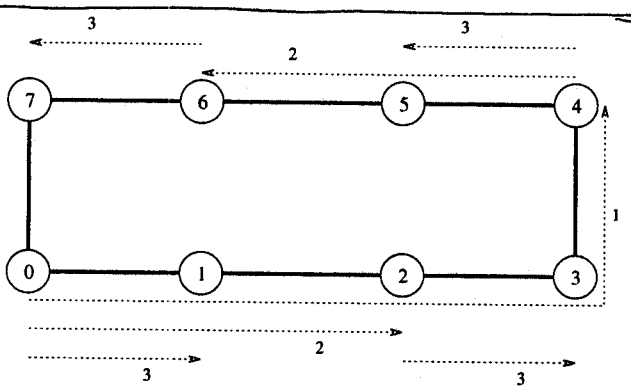


Figure 3.6 One-to-all broadcast with CT routing on an eight-processor ring.

- map HC algo to ring

- i^{th} step: each processor that has data sends it to a processor at a distance $p/2^i$; all msgs in same direction

$$T^{CT}_{1\to all} = \sum_{i=1}^{\log p} \left(t_s + t_w m + t_h \frac{p}{2^i}\right) = (t_s + t_w m)\log p + t_h(p-1)$$

For large m, t_h term becomes insignificant

⇒ CT reduces comm. time over SF by factor of $\frac{p}{\log p}$

- apply ring algorithm to each dimension serially

$$T_{1\to all} = (t_s + t_w m)\log p + 2t_h(\sqrt{p}-1)$$

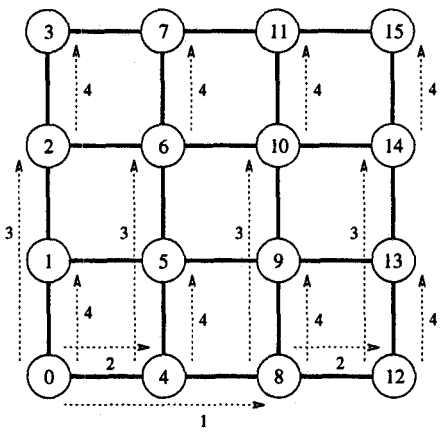


Figure 3.7 One-to-all broadcast on a 16-processor square mesh with CT routing.

- Map HC algorithm to tree
- No congestion
- Different # of switching nodes along different paths

$$T^{tree}_{1\to all} = \left(t_s + t_w m + t_h(\log p + 1)\right) \log p$$

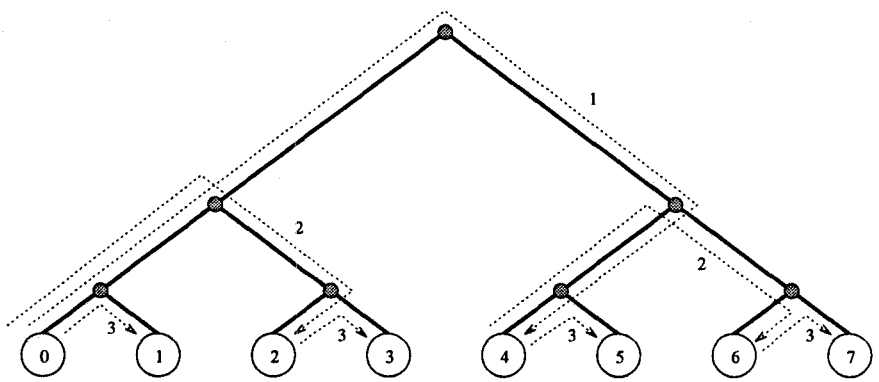


Figure 3.8 One-to-all broadcast on an eight-processor tree.

All→All BC, Reduction & Prefix Sums

- All→All BC used in – matrix-matrix, matrix-vector multiplication
  – other matrix operations
- Dual: multi-node accumulation

  (Recall: single-node acc: each proc has diff data; all data combined at single proc. using associative op. (eg +, *, min, max, logical bit-op etc)
- p 1→all BCs simultaneous: msgs traversing same path in same step are concatenated

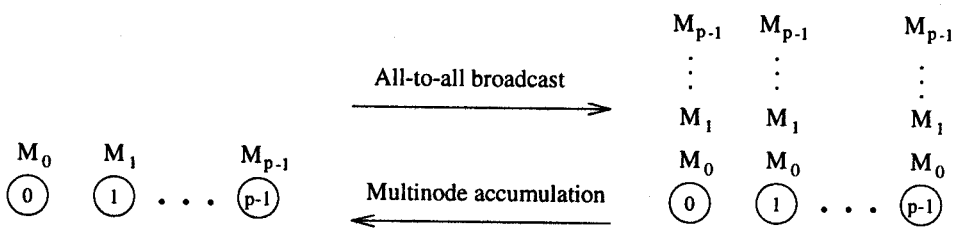


Figure 3.9 All-to-all broadcast and multinode accumulation.

- all channels kept busy simultaneously
- each proc has info to send
- PIPELINING
- $T_{all \to all} = (t_s + t_w m)(p-1)$

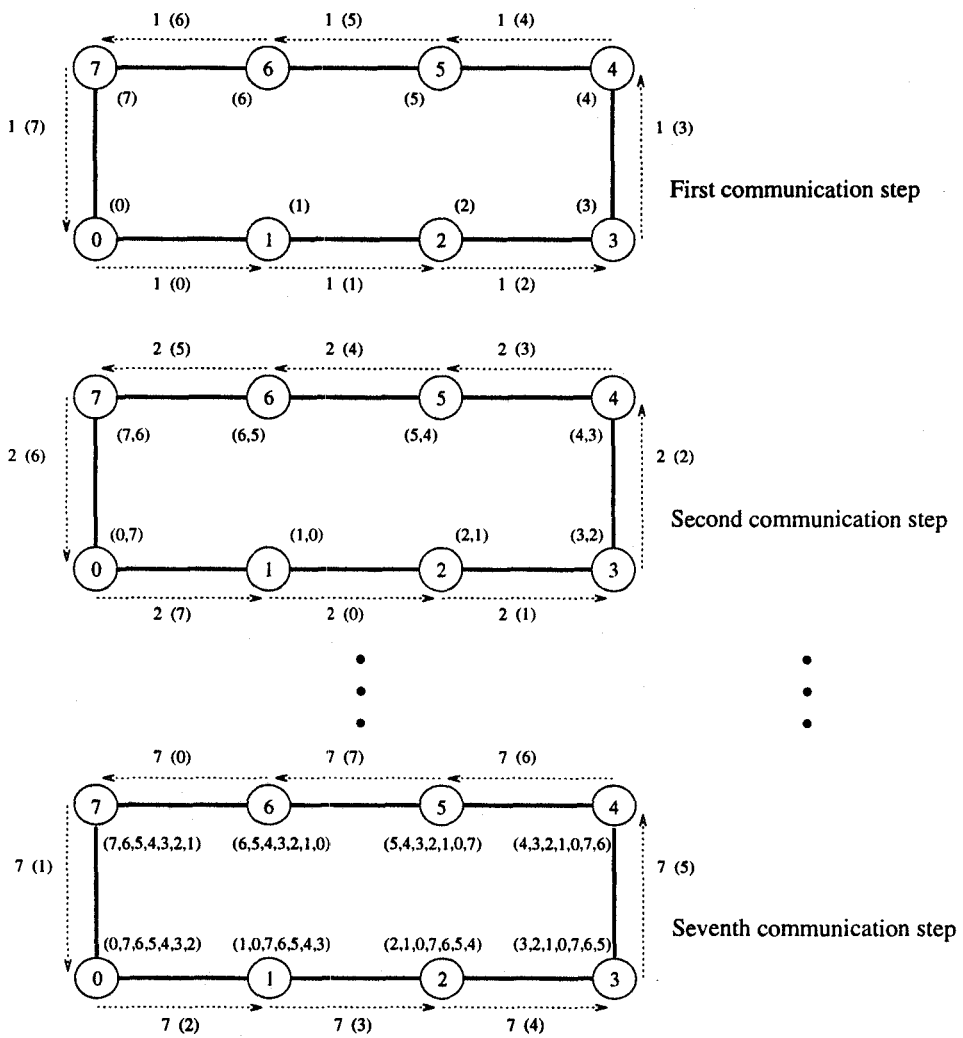
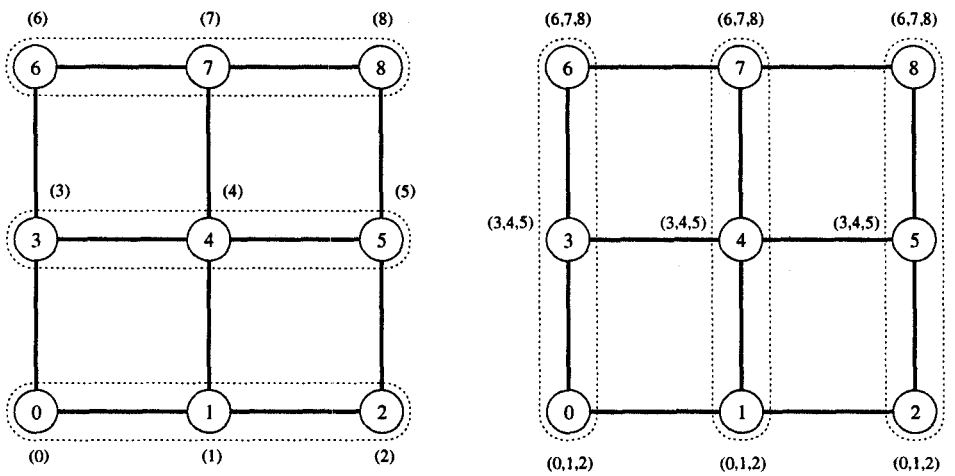


Figure 3.10 All-to-all broadcast on an eight-processor ring with SF routing. In addition to the time step, the label of each arrow has an additional number in parentheses. This number labels a message and indicates the processor from which the message originated in the first step. The number(s) in parentheses next to each processor are the labels of processors from which data has been received prior to the communication step. Only the first, second, and last communication steps are shown.

[Note: pipelined broadcasts useful in parallel algos such as Gaussian elimination, back substitution, mat. mult, shortest path (Floyd's algo) etc.]

- Phase 1 (row): $(t_s + t_w m)(\sqrt{p} - 1)$
- Phase 2 (col): $(t_s + t_w m\sqrt{p})(\sqrt{p} - 1)$



(a) Initial data distribution

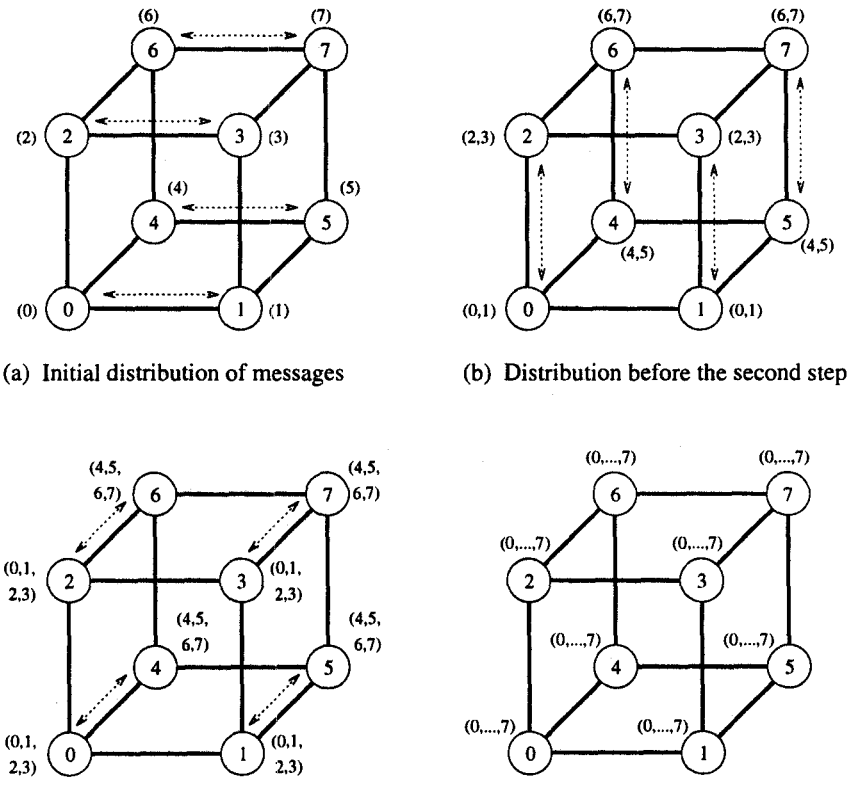
(b) Data distribution after rowwise broadcast

Figure 3.11 All-to-all broadcast on a 3×3 mesh. The groups of processors communicating with each other in each phase are enclosed by dotted boundaries. By the end of the second phase, all processors get (0,1,2,3,4,5,6,7) (that is, a message from each processor).

- $\log p$ steps
- i^{th} step msg size $= 2^{i-1} \times m$
- $T_{all \to all} = \sum_{i=1}^{\log p} (t_s + 2^{i-1} t_w m) = t_s \log p + t_w m (p-1)$
- Reduction: each proc. starts w/value; needs to know sum of all
  : (used to implement barrier synchronization)

In algorithm below, add instead of concat at each step

$T_{reduction} = (t_s + t_w) \log p$



(a) Initial distribution of messages

(b) Distribution before the second step

(c) Distribution before the third step

(d) Final distribution of messages

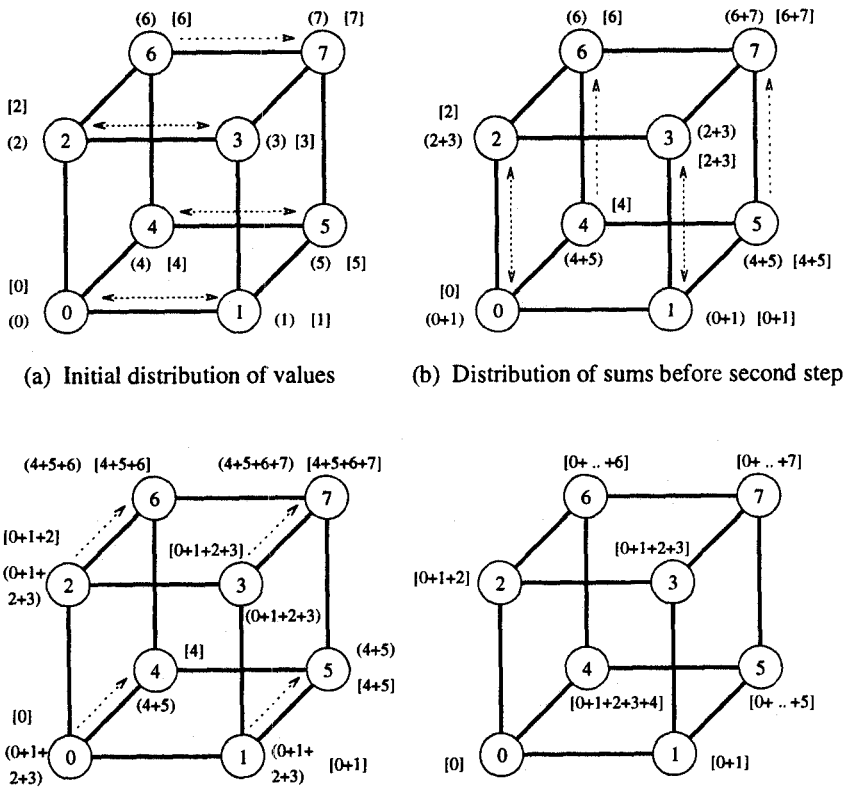
Figure 3.12 All-to-all broadcast on an eight-processor hypercube.


```
1.  procedure ALL_TO_ALL_BC_HCUBE(my_id, my_msg, d, result)
2.  begin
3.      result := my_msg;
4.      for i := 0 to d - 1 do
5.      begin
6.          partner := my_id XOR 2^i;
7.          send result to partner;
8.          receive msg from partner;
9.          result := result ∪ msg;
10.     endfor;
11. end ALL_TO_ALL_BC_HCUBE
```

Program 3.6 All-to-all broadcast on a d-dimensional hypercube.

• PREFIX SUMS: Proc k has n_k; computes $s_k = \sum_{i=0}^{k} n_i$

→ P_k uses info only from k-processor subset of those procs whose labels $\leq k$



(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

Figure 3.13 Computing prefix sums on an eight-processor hypercube. At each processor, square brackets show the local prefix sum accumulated in a buffer and parentheses enclose the contents of the outgoing message buffer for the next step. (Not all msgs are shown).


```
1.  procedure PREFIX_SUMS_HCUBE(my_id, my_number, d, result)
2.  begin
3.      result := my_number;
4.      msg := result;
5.      for i := 0 to d - 1 do
6.      begin
7.          partner := my_id XOR 2^i;
8.          send msg to partner;
9.          receive number from partner;
10.         msg := msg + number;
11.         if (partner < my_id) then result := result + number;
12.     endfor;
13. end PREFIX_SUMS_HCUBE
```

Program 3.7 Prefix sums on a d-dimensional hypercube.

CUT-Through Routing

- 1→all : $HC^{SF}(= HC^{CT})$ mapped to ring & mesh gave better ring & mesh algos for CT
- all→all : mapping HC algo to ring & mesh is not strictly better because of congestion (see below)
- all→all : time$\propto [t_w m(p-1)]$ earlier
  : this is <u>lower bound</u> if a processor can communicate on only 1 port at a time

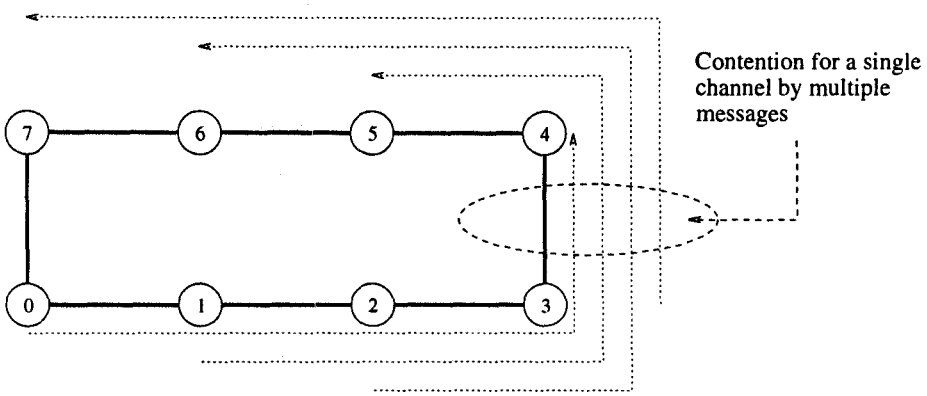


Figure 3.14 Contention for a channel when the communication step of Figure 3.12(c) for the hypercube is mapped onto a ring.