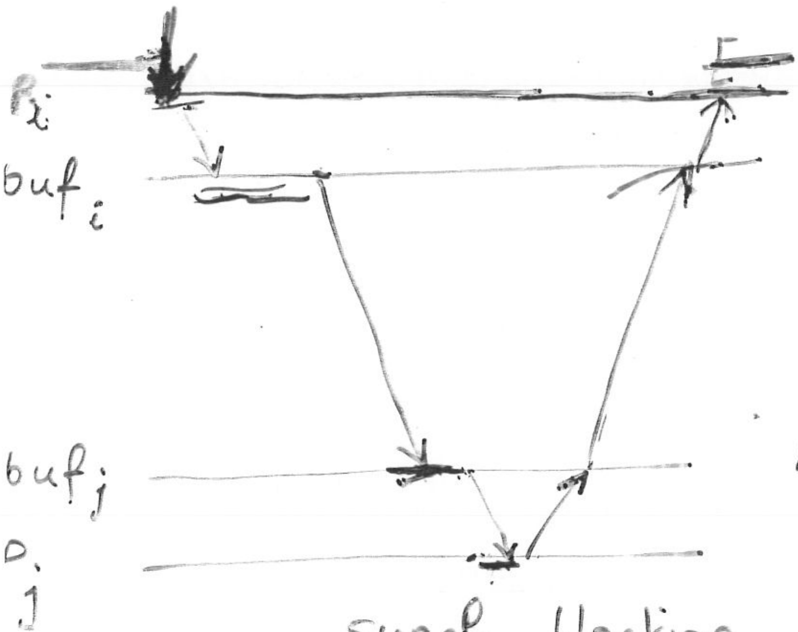
Nonblocking
vs
Send : Blocking : program blocks until command executes fully

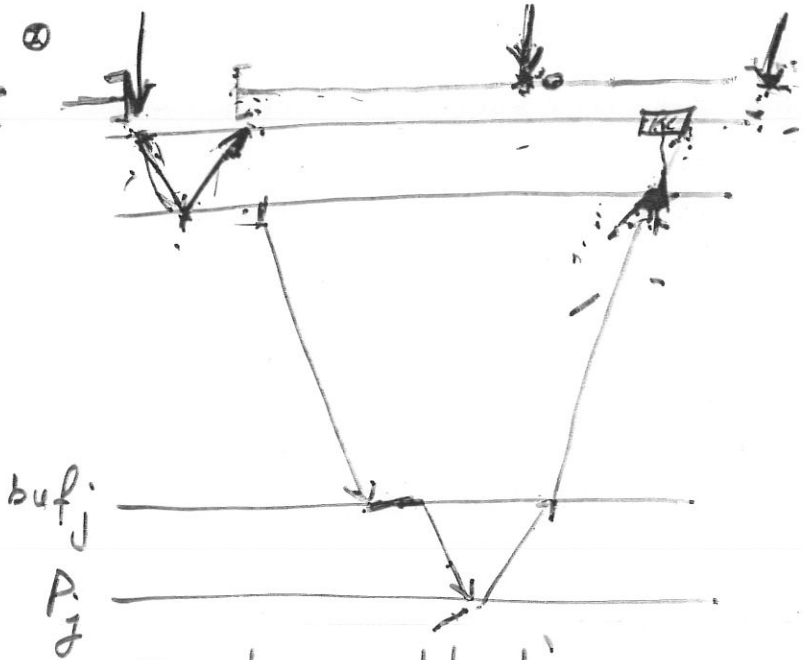
Synchronous : send & receive must handshake
vs
Asynchronous

Receive : always Synchronous
Nonblocking
vs
Blocking

MPI_Init	Initialize
MPI_Finalize	Terminate
MPI_Comm_Size	Determines # processes
MPI_Comm_rank	Determines label of calling process
MPI_Send	Sends msg
MPI_Recv	Receives msg
MPI_Bcast	1 → all BC
MPI_Reduce	All → 1 reduction
MPI_Allgather	All → all BC
MPI_Reduce_scatter	All → all reduction
MPI_Allreduce	All reduce
MPI_Gather	Gather
MPI_Scatter	Scatter
MPI_Alltoall	All → all personalized



synch-blocking

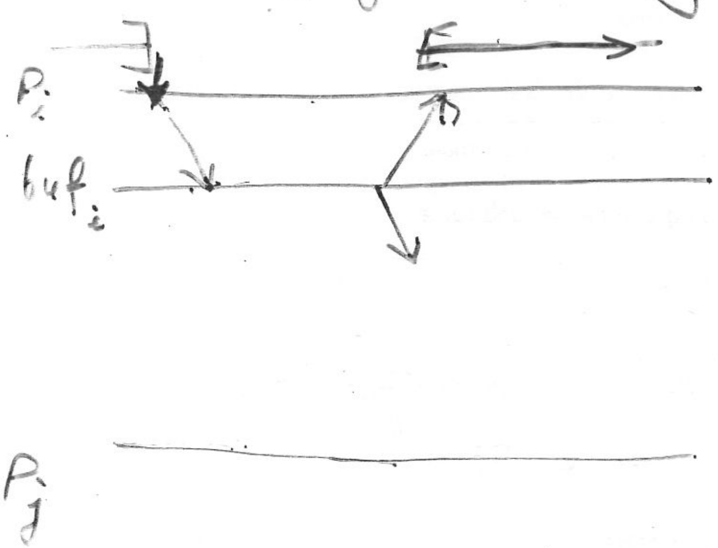


synch-nonblocking

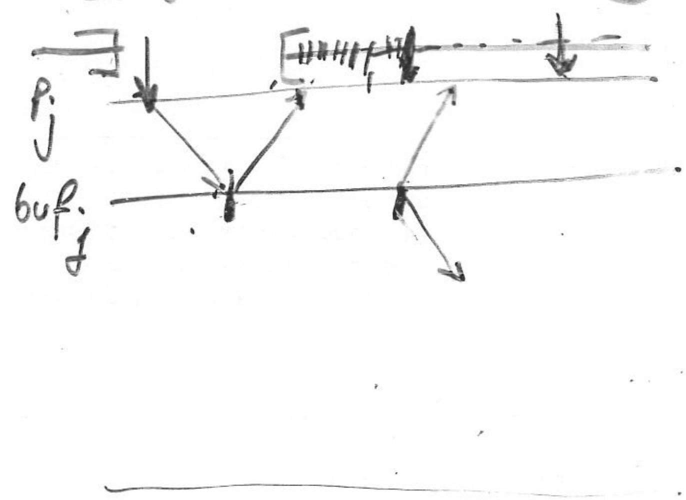
types of send ops

send(X, ---, (handle))
wait (handle)

asynch-blocking



asynch-nonblocking



- Synchronous vs asynchronous execution
- Communication synchrony ; processor synchrony
- ~~Event~~ types

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
- Approx. for transfer time: $t_s + t_w m$?

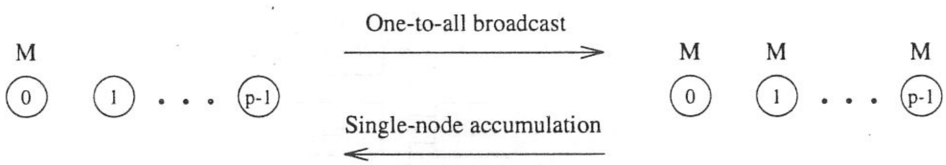


Figure 3.1 One-to-all broadcast and single-node accumulation. [duals]

- Fig 3-1

  $1 \to$ all broadcast & single-node accumulation:

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

- $1 \to$ all broadcast (MPI_Bcast) (MPI_Reduce) • circular shift
- all $\to$ all broadcast, reduction, prefix sums (MPI_Allgather) (MPI_Reduce_Scatter)
- ~~$1 \to$ all personalized communication~~
- all $\to$ all personalized communication (MPI_Alltoall)
- MPI Gather MPI Scatter

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

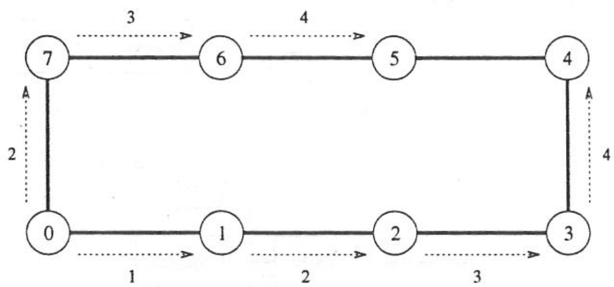
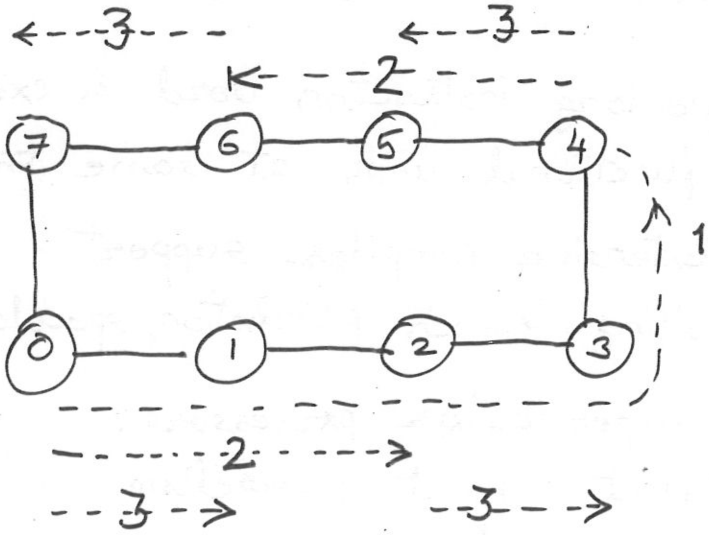


Figure 3.2 One-to-all broadcast on an eight-processor ring with SF routing. Processor 0 is the source of the broadcast. Each message transfer step is shown by a numbered, dotted arrow from the source of the message to its destination. The number on an arrow indicates the time step during which the message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



- $T = \log p \left[t_s + t_w m \right] + t_h \times p.$

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

$$\underset{A}{\begin{bmatrix} & \\ & \end{bmatrix}} \times \underset{B}{\begin{bmatrix} \vdots \end{bmatrix}} = \underset{C}{\begin{bmatrix} \vdots \end{bmatrix}}$$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & & & \vdots \\ \vdots & & & \vdots \\ a_{41} & & & a_{44} \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} =$$

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

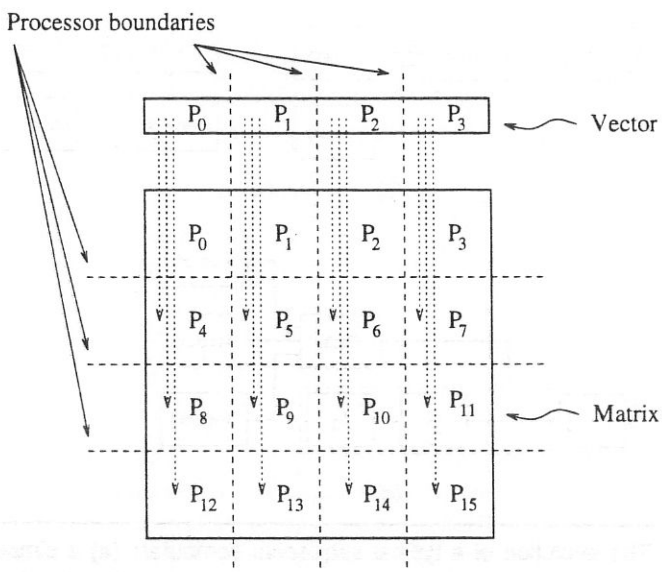


Figure 3.3 One-to-all broadcast in the multiplication of a 4×4 matrix with a 4×1 vector.

- All-to-one reduction rowwise to calculate row element of product vector

① $1 \rightarrow$ all broadcast in row

② $1 \rightarrow$ all broadcast in columns in parallel

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

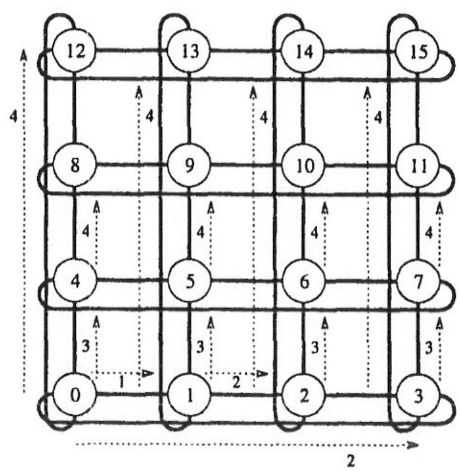


Figure 3.4 One-to-all broadcast on a 16-processor mesh with SF routing. Copyright (r) 1994 Benjamin/Cummings Publishing Co.

For a 3-D mesh,

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

```
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 */
   begin
      mask := mask XOR 2^i;  /* Set bit i of mask to 0 */
      if (my_id AND mask) = 0 then
      /* If the lower i bits of my_id are 0 */
         if (my_id AND 2^i) = 0 then
         begin
            msg_destination := my_id XOR 2^i;
            send X to msg_destination;
         endif
         else
         begin
            msg_source := my_id XOR 2^i;
            receive X from msg_source;
         endelse;
   endfor;
end ONE_TO_ALL_BC
```

111 ⊕ 100 — Iter. #1

011 ⊕ 010 — Iter #2

001 ⊕ 001 — Iter #3

000

phases on a
dimension
imensional
mesh with
ded for the
dimension

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.

- 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)

for $i = 2$, $2^i = 100$
$i = 1$, $2^i = 010$
$i = 0$, $2^0 = 001$

- '0' in bit posn i: $\Rightarrow$ send
- '1' in bit posn i: $\Rightarrow$ receive

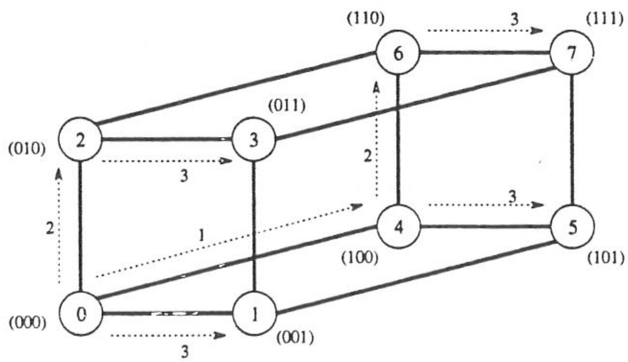


Figure 3.5 One-to-all broadcast on a three-dimensional hypercube. The binary representations of processor labels are shown in parentheses.

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

Which processes are active in iteration k?

$$\left.\begin{array}{lll} k=1 & : & x\ 0\ 0 \cdots 0 \\ k=2 & : & x\ x\ 0\ 0 \cdots 0 \\ k=3 & : & x\ x\ x\ 0 \cdots 0 \\ \vdots & & \quad\vdots \\ k=d & & x\ x\ x\ x \cdots x \end{array}\right\} \Rightarrow 0^s \text{ in the } (d-k) \text{ LSB}^s$$

i.e., $\text{my-id} \overset{\text{AND}}{} \underbrace{\underbrace{0\cdots0}_{k}\ \underbrace{1\cdots1}_{d-k}}_{\text{MASK}} = 0\cdots00\cdots0$

How to generate MASK? Initialize to $2^d - 1$

Then: $\text{MASK} \leftarrow \text{MASK} \oplus 2^{d-k}$

$$\left.\begin{array}{ll} \text{Init:} & 1111 \text{---} 1111 \\ \text{Iter 1:} & 0111 \text{----} 1111 \\ \text{Iter 2:} & 0011 \text{----} 1111 \\ & \vdots \end{array}\right\} \leftarrow$$

Variable substitution: $i = d-k$.

Thus, active if $\text{my-id} \overset{\text{AND}}{} \text{MASK} = 0 \cdots 0$, where $\text{MASK} = \text{MASK} \oplus 2^i$

If active, send if $\text{my-id} \overset{\text{AND}}{} 2^{d-k} = 0$ 'o' in bit position i

i.e. $\text{my-id} \overset{\text{AND}}{} 2^i = 0$

(otherwise receive)

```
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 */
   begin
      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
         begin
            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 */
         endif
         else
         begin
            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
```

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

```
procedure SINGLE_NODE_ACC(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
   begin /* Select processors whose lower i bits are 0 */
      if (my_id AND mask) = 0 then
         if (my_id AND 2^i) ≠ 0 then
         begin
            msg_destination := my_id XOR 2^i;
            send sum to msg_destination;
         endif
         else
         begin
            msg_source := 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 SINGLE_NODE_ACC
```

Program 3.3 Single-node accumulation on a d-dimensional hypercube. Each processor contributes a msg X containing m words, & processor Ø 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

    $\Rightarrow$ 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, ~~over~~ instead of SF

CUT-THROUGH ROUTING

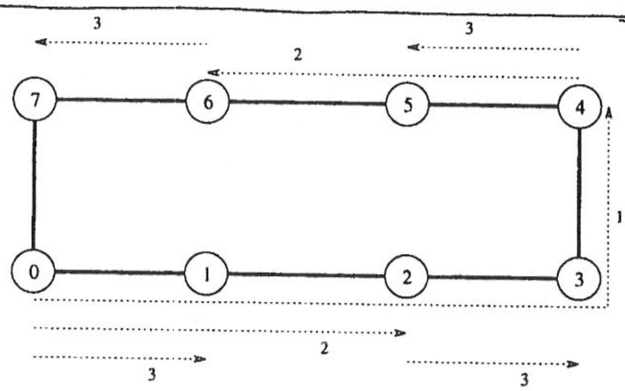


Figure 3.6 One-to-all broadcast with CT routing on an eight-processor ring.

- 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

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

• apply ring algorithm to each dimension serially

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

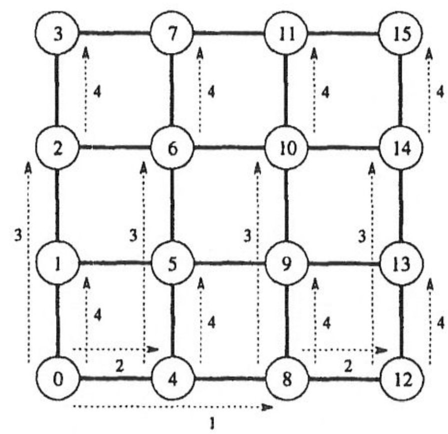


Figure 3.7 One-to-all broadcast on a 16-processor square mesh with CT routing.

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

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

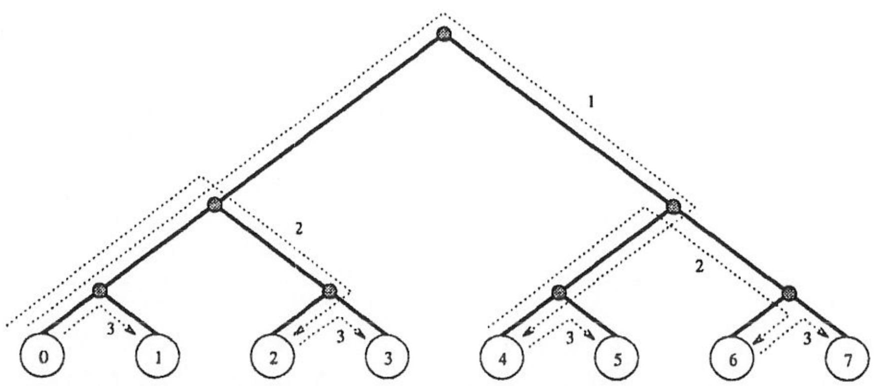


Figure 3.8 One-to-all broadcast on an eight-processor tree.

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

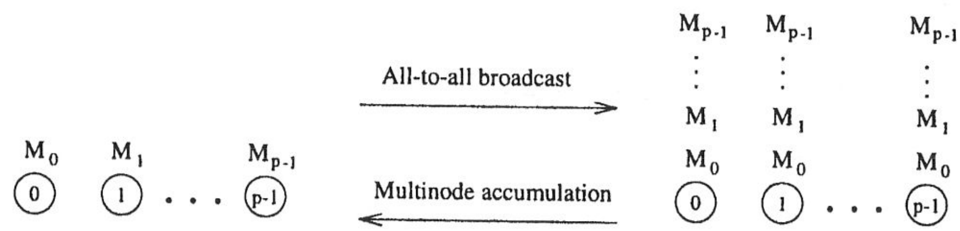


Figure 3.9 All-to-all broadcast and multinode accumulation.

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

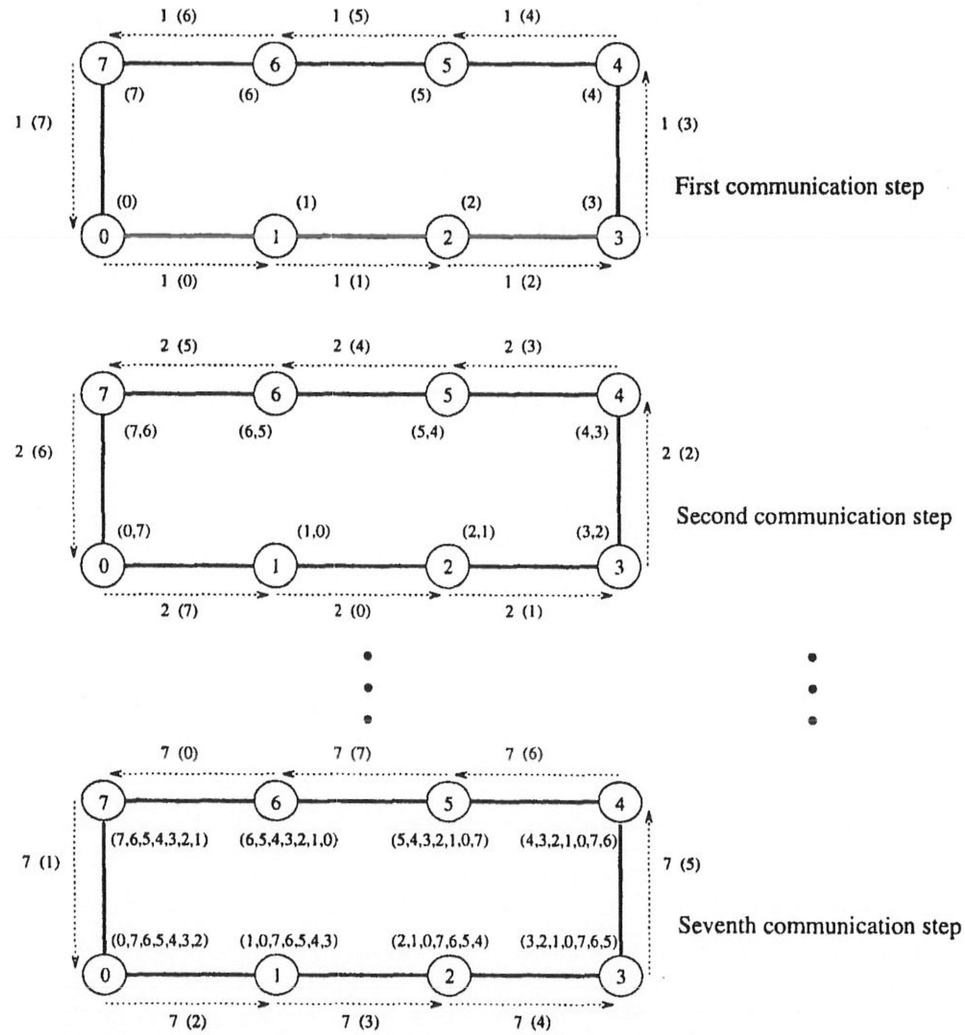


Figure 3.10 All-to-all broadcast on an eight-processor ring with $\boxed{\text{SF}}$ routing. In addition to the time step, the label of each arrow has an additional number in parentheses. This number labels a message and indicates the processor from which the message originated in the first step. The 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.

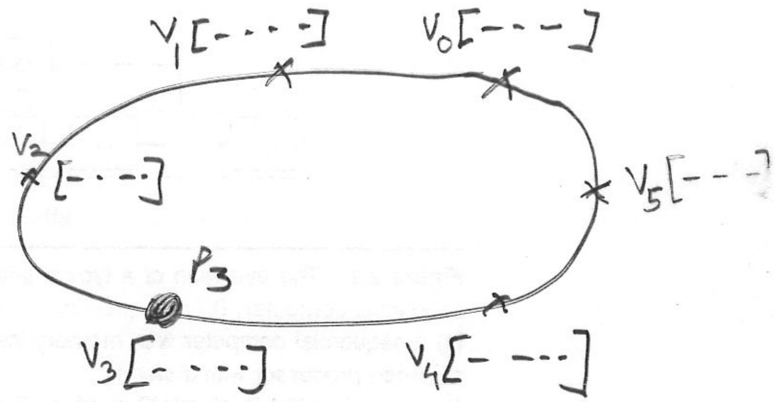
[Note: pipelined broadcasts useful in parallel algos such as Gaussian elimination, back substitution, mat. mult, shortest path (Floyd's algo) etc.]

```
proc ALL_TO_ALL_BC_RING (my-id, my-msg, p, result)

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
    rcv msg from left
    result = result U msg
```

```
proc ALL-TO-ALL-RED-RING (my-id, my-msg, p, result)

recv = 0
for i = 1 to p-1 do
        j = (my-id + i) mod p
        Temp = mymsg [j] + recv
        send Temp to left
        rcv recv from right
result = my_msg [my-id] + recv
```



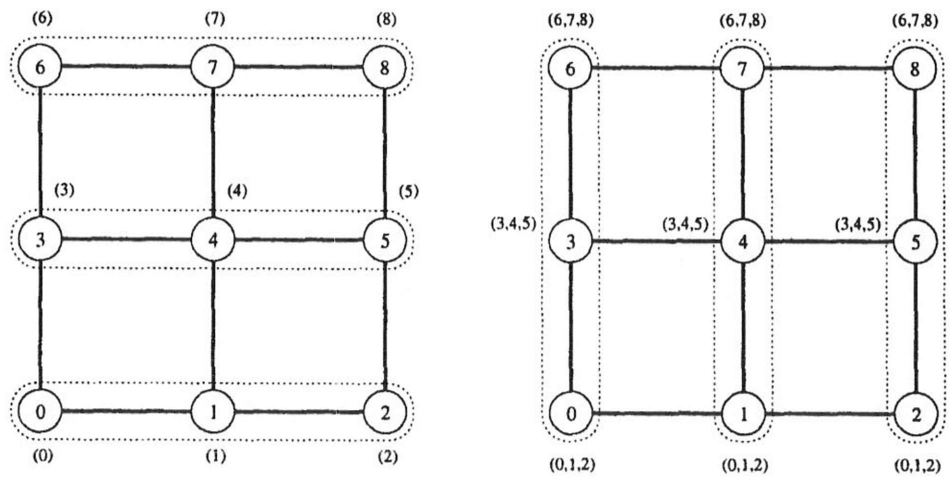
eg $\underline{P_3}$

$i = 1$: send $V_3[4]$ to P_2
rcv $V_4[5]$ from P_4

$i = 2$: send $(V_3[5] + V_4[5])$ to P_2
rcv $(V_4[0] + V_5[0])$ from P_4

$i = 3$: send $(V_3[0] + V_4[0] + V_5[0])$ to P_2
rcv $(V_4[1] + V_5[1] + V_0[1])$ from P_4

- 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

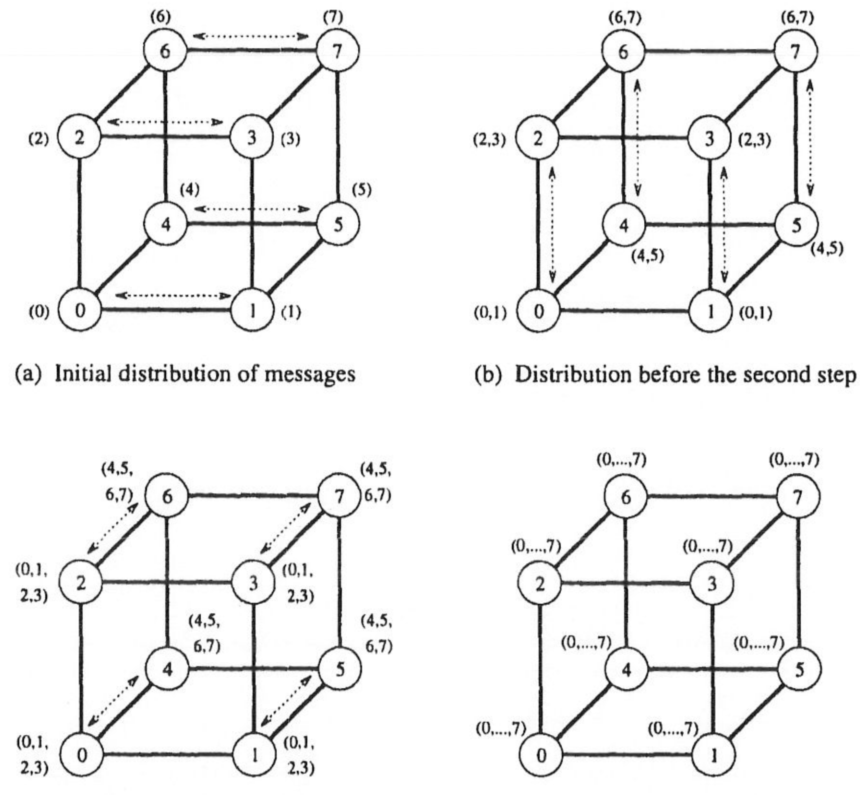
Figure 3.11 All-to-all broadcast on a 3 × 3 mesh. The groups of processors communicating with each other in each phase are enclosed by dotted boundaries. By the end of the second phase, all processors get (0,1,2,3,4,5,6,7) (that is, a message from each processor).

- $\log p$ steps
- i^{th} step msg size $= 2^{i-1} \times m$
- $T_{all \to all} = \sum_{i=1}^{\log p} (t_s + 2^{i-1} t_\omega m) \Rightarrow t_s \log p + t_\omega m (p-1)$

- Reduction: each proc. starts w/value; needs to know sum of all
  : (used to implement barrier synchronization)

In algorithm below, <u>add</u> instead of <u>concat</u> at each step

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



(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.


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

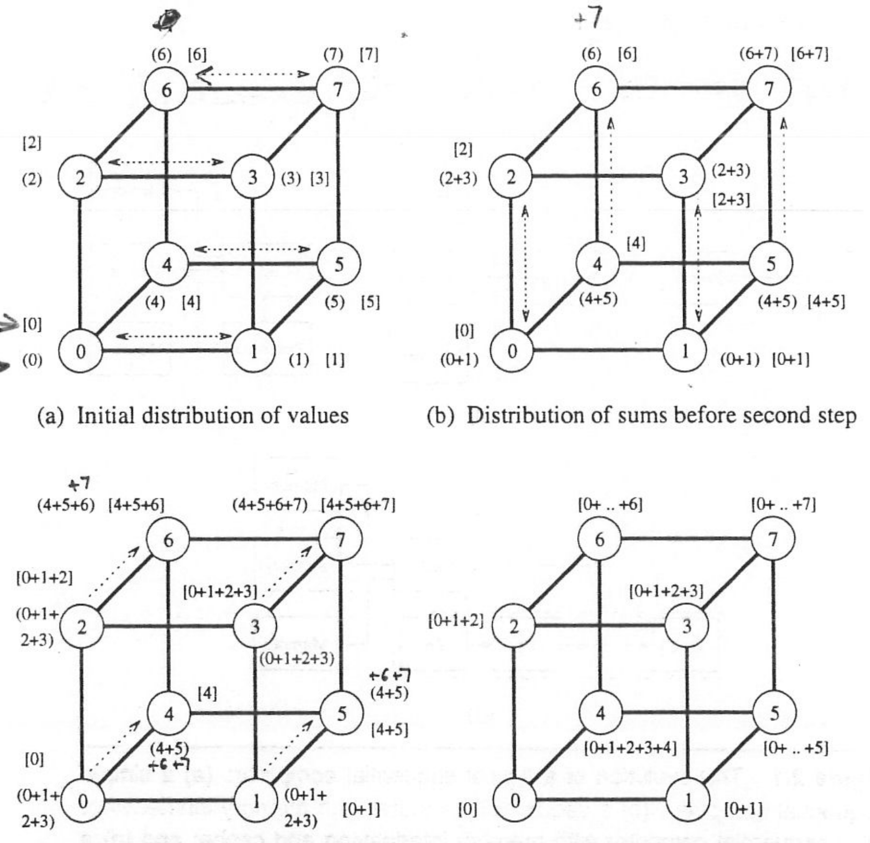
Program 3.6 All-to-all broadcast on a d-dimensional hypercube.

• 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 $\leq k$

result (local prefix sum)

msg



(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).


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

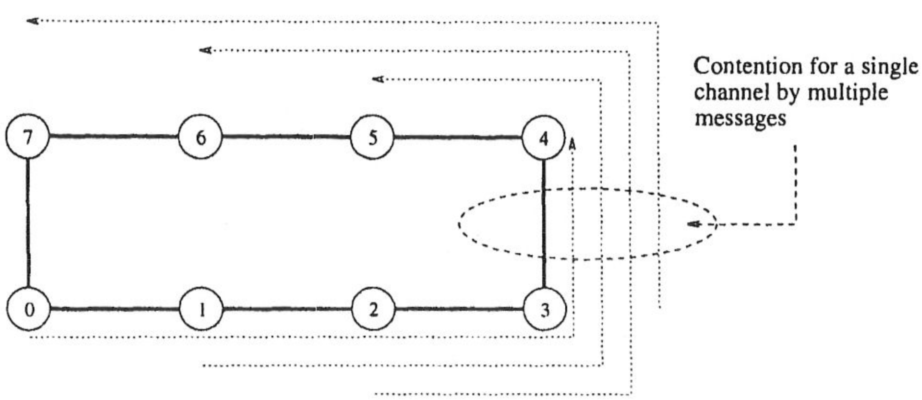
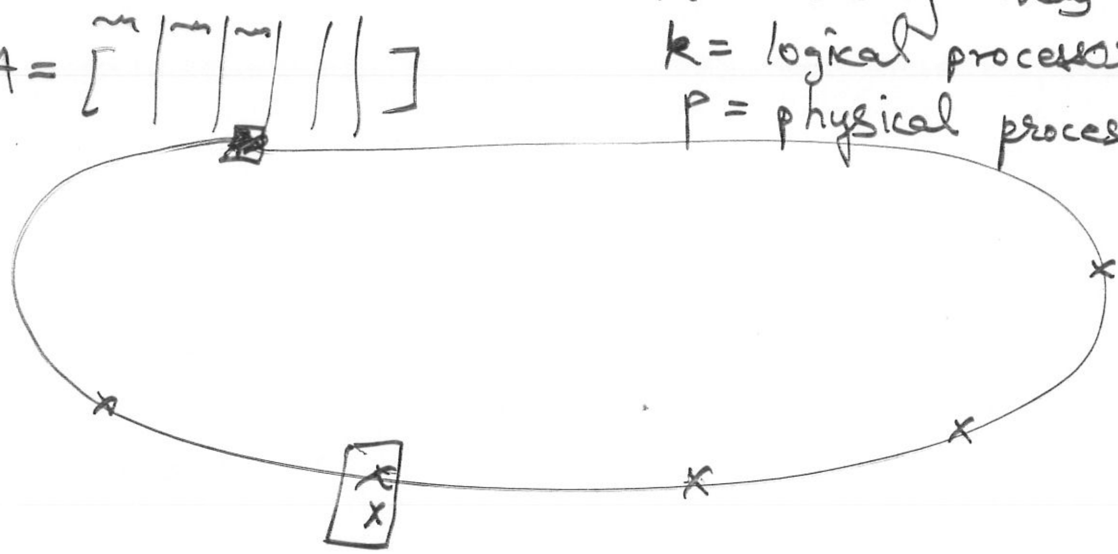


Figure 3.14 Contention for a channel when the communication step of Figure 3.12(c) for the hypercube is mapped onto a ring.

n = size of array A
k = logical processors
p = physical processors.

A =



$n = 100{,}000$

$k = 500$

200 elems/logical proc.

$p = 10$

ONE → ALL PERSONALIZED communication (single node SCATTER)

- single processor sends a unique msg to every other processor
- Dual: single node GATHER
  - single proc. collects a unique msg from each other proc.
  - <u>Note</u>: GATHER differs from Accumulation !
- Complexity (1 → all personalized) ≡ complexity (all → all BC)

source ~~each~~ proc sends $m(p-1)$ each proc receives $m(p-1)$

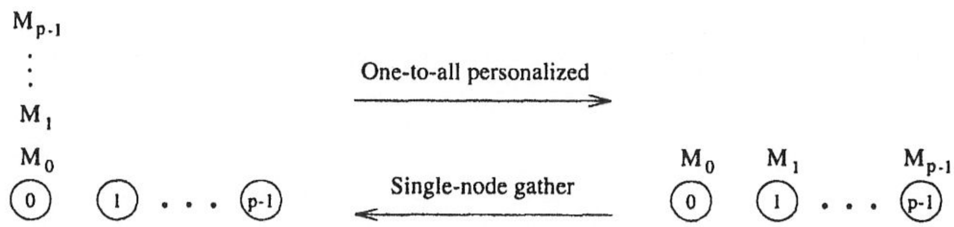
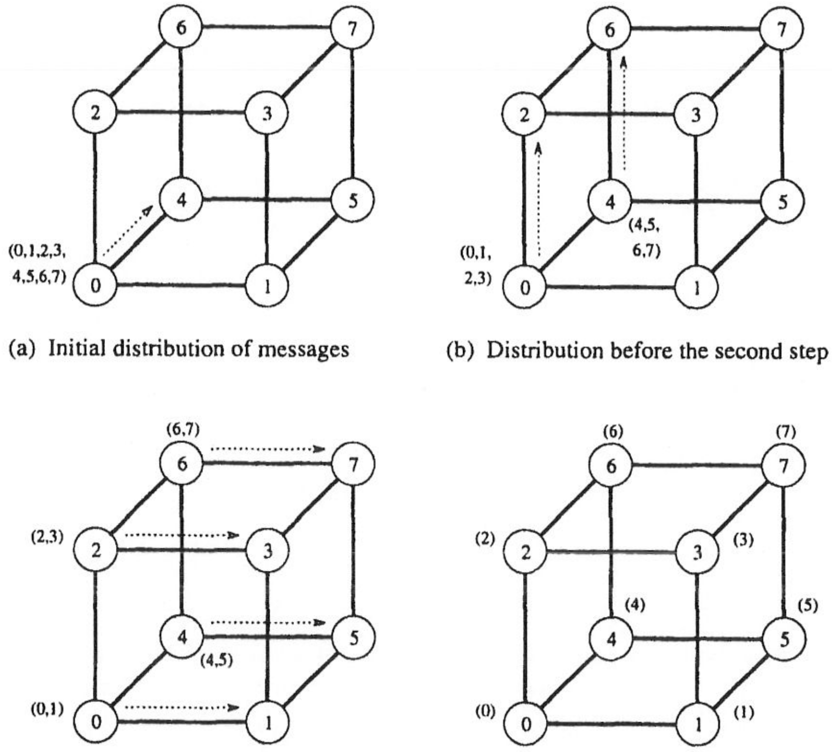


Figure 3.15 One-to-all personalized communication and its dual—single-node gather.

- $\log p$ steps
- same communication pattern as for $1 \to$ all BC
  but msg size & contents are different
- $T_{all \to all (pers)} = \sum_{i=1}^{\log p} (t_s + 2^{i-1} t_w m) = t_s \log p + t_w m (p-1)$



(a) Initial distribution of messages
(b) Distribution before the second step
(c) Distribution before the third step
(d) Final distribution of messages

Figure 3.16 One-to-all personalized communication on an eight-processor hypercube. Msgs are labeled by the labels of their dests

- Ring (CT & SF)

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

- 2D square mesh (CT & SF)

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

ALL → ALL PERSONALIZED COMMUNICATION

- each proc sends distinct msg to each other proc
- eg uses: parallel FFT, matrix transpose, parallel database join op
- Communication pattern same as for all → all BC
  → msg size & contents differ

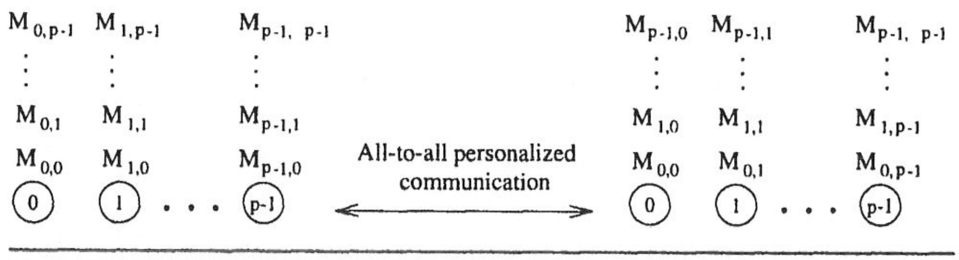


Figure 3.17 All-to-all personalized communication.

- Ring (SF)

→ $(p-1)$ steps

→ i^{th} step msg size $= m(p-i)$

$$T_{\text{all}\to\text{all (pers)}} = \sum_{i=1}^{p-1} \left(t_s + t_w m(p-i)\right) = \left(t_s + \tfrac{1}{2} t_w m p\right)(p-1)$$

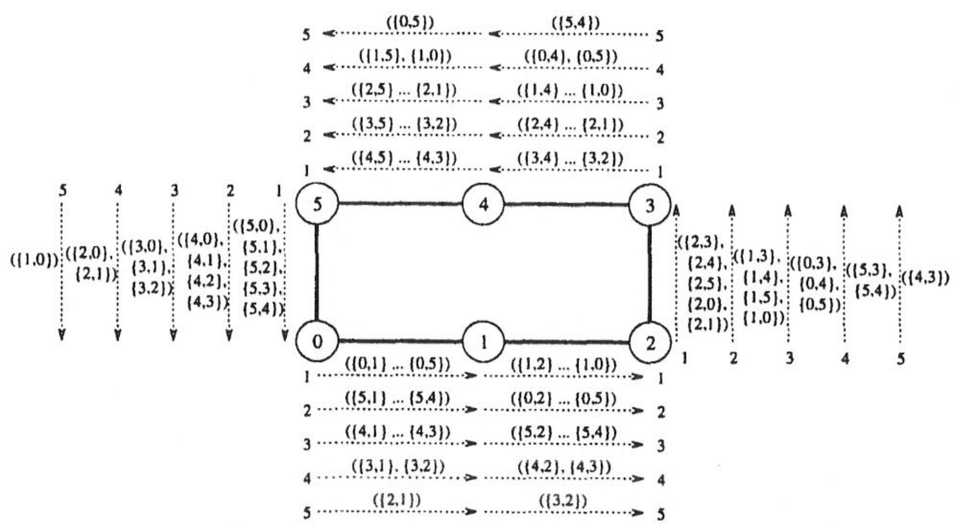


Figure 3.18 All-to-all personalized communication on a six-processor ring. The label of each message is of the form $\{x, y\}$, where x is the label of the processor that originally stored the message, and y is the label of the processor that is the final destination of the message. The label $(\{x_1, y_1\}, \{x_2, y_2\}, \ldots, \{x_n, y_n\})$ indicates a message that is formed by concatenating n individual messages.

→ assumption: msgs sent in 1 direction only

- Total traffic $= m(p-1) \times \frac{p}{2} \times p$, shared across p channels
- Communic time $\geq \frac{t_w m p(p-1)}{2}$ (same as above for SF)

∴ cannot be improved with CT

- Phase 1 (row): $\left(t_s + \frac{t_w (m\sqrt{p})\sqrt{p}}{2}\right)(\sqrt{p}-1)$ — msg size, #nodes
- Phase 2 (col): same

MESH (SF)

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

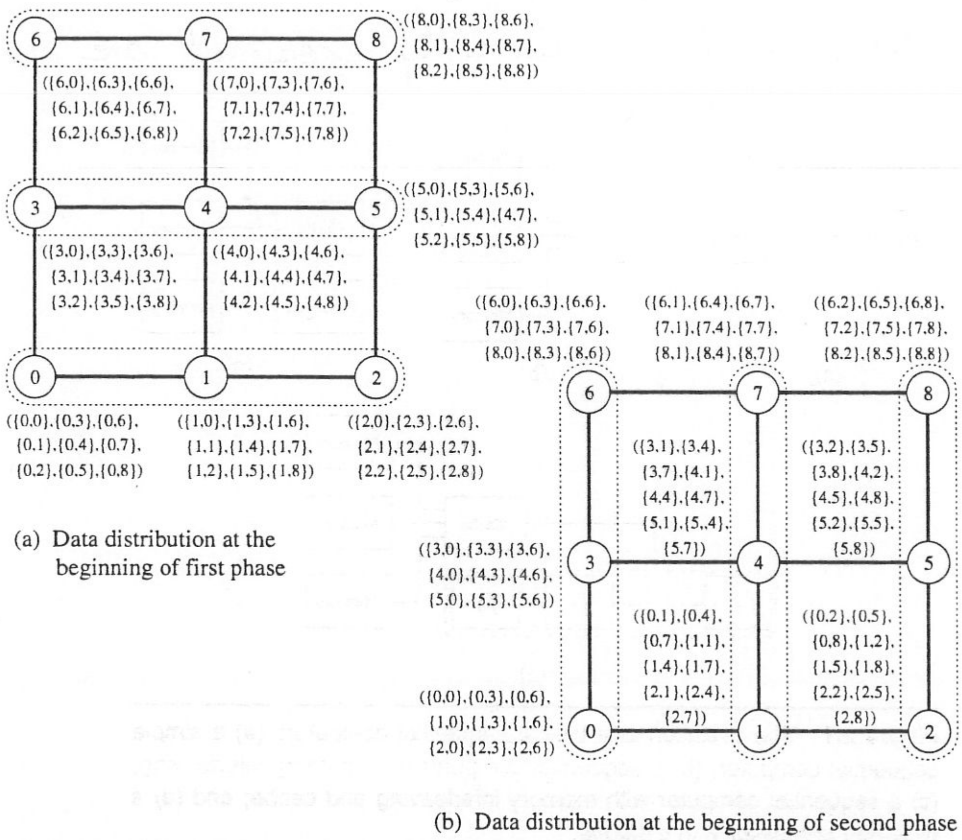
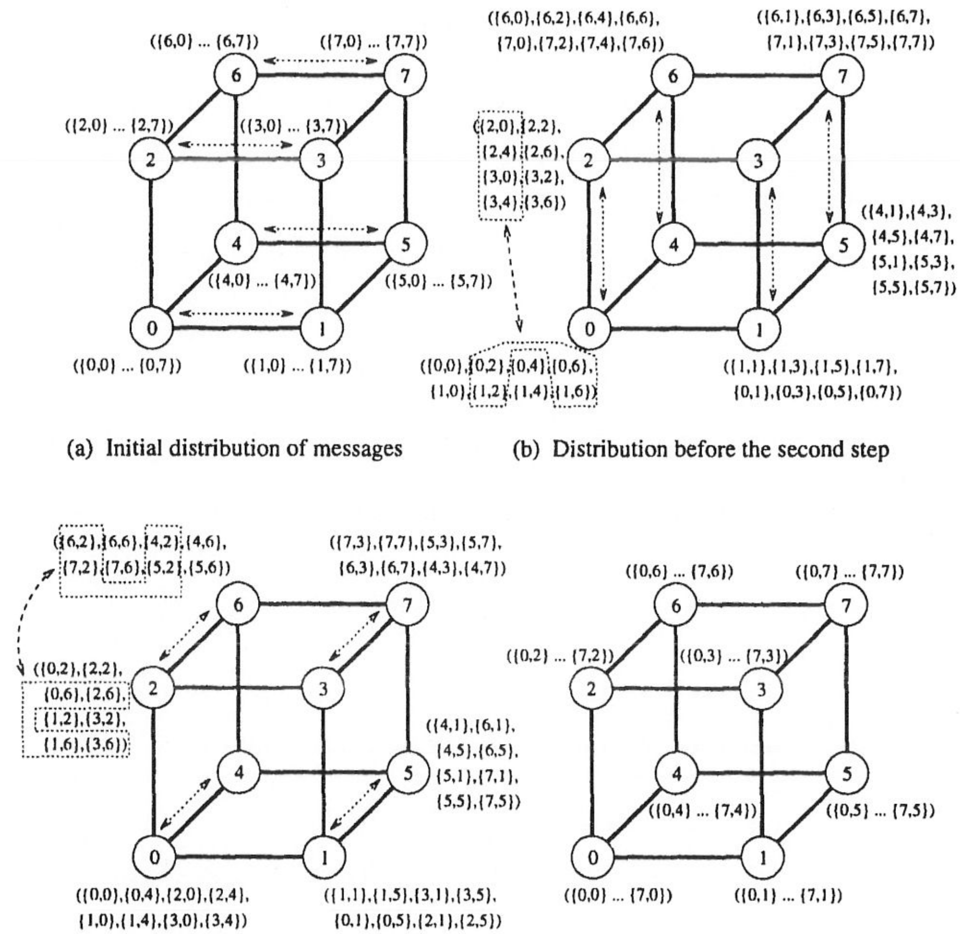


Figure 3.19 The distribution of messages at the beginning of each phase of all-to-all personalized communication on a 3 × 3 mesh. At the end of the second phase, processor i has messages $(\{0,i\}, \ldots, \{8,i\})$, where $0 \le i \le 8$. The groups of processors communicating together in each phase are enclosed in dotted boundaries.

- Extra ovhd for local rearrangement of data
  → $t_r m p$, where t_r = time to do a read & write on single word

- Cannot be improved by CT (same reasoning as for ring)

- Extend 2D to log p dimensions
- Send data for the "other" subcube, in each step
- $\frac{mp}{2}$ words exchanged along bi-directional links in each of log p steps
- $T_{all \to all\,(pers)} = (t_s + \frac{1}{2} t_w m p) \log p$



(a) Initial distribution of messages (b) Distribution before the second step

(c) Distribution before the third step (d) Final distribution of messages

Figure 3.20 All-to-all personalized communication on a three-dimensional hypercube with SF routing.

- local rearrangements ovhd $= t_r m p \log p$

  (t_r = time to do a single read and write on a word)

• lower bound $T_{all \to all\ (pers)} = \dfrac{t_w m (p-1)(p \log p)/2}{(p \log p)/2} = t_w m (p-1)$

$\boxed{\text{CT}}$

• $(p-1)$ min. communication steps; j^{th} step: i exchanges data w/ $(i \text{ XOR } j)$

• no congestion

• Ecube routing time/step $= t_s + t_w m + l t_h \Longrightarrow$ total $T_{all \to all\ (pers)} =$

$$(t_s + t_w m)(p-1) + \frac{t_h\, p \log p}{2}$$

(t_h term is greater than for SF)

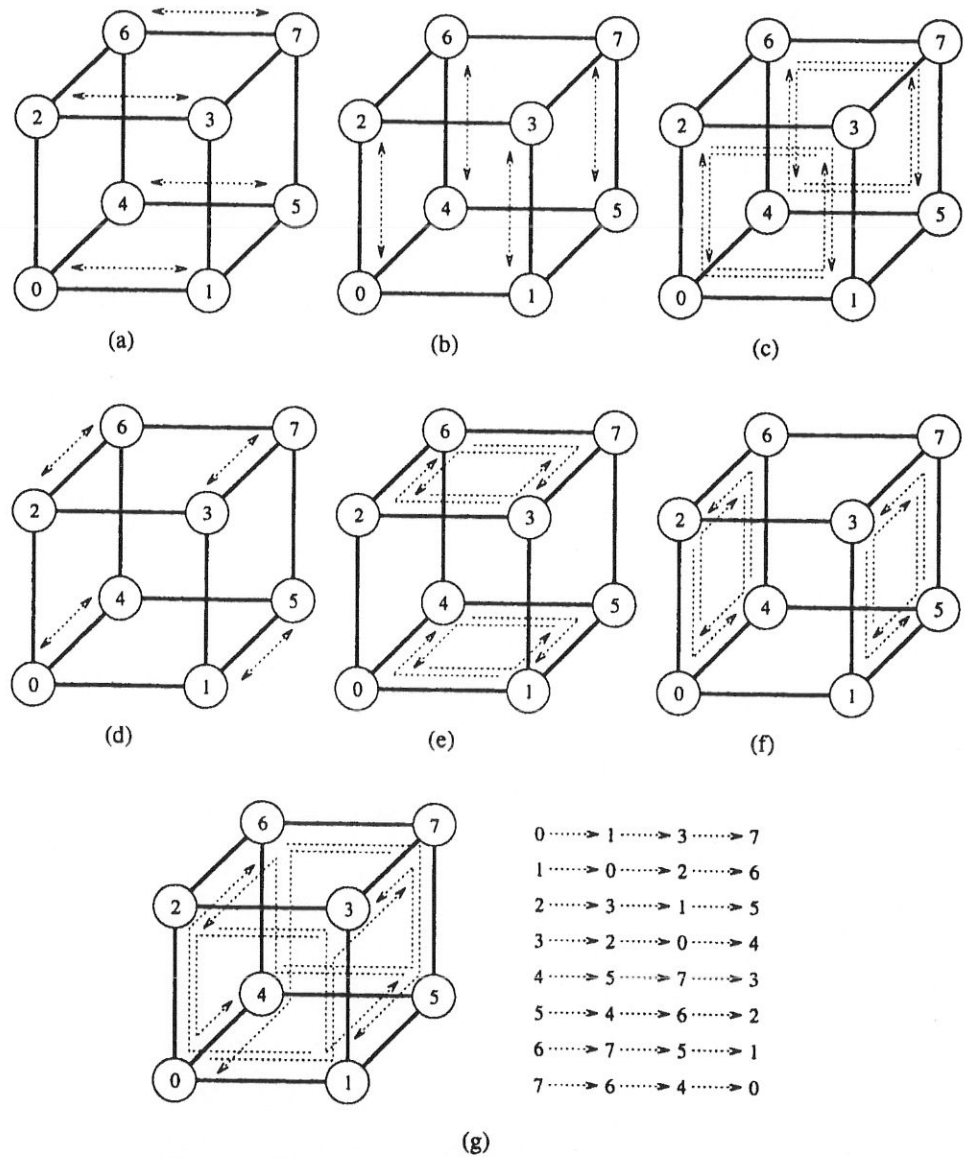


Figure 3.21 Seven steps in all-to-all personalized communication on an eight-processor hypercube with CT routing.

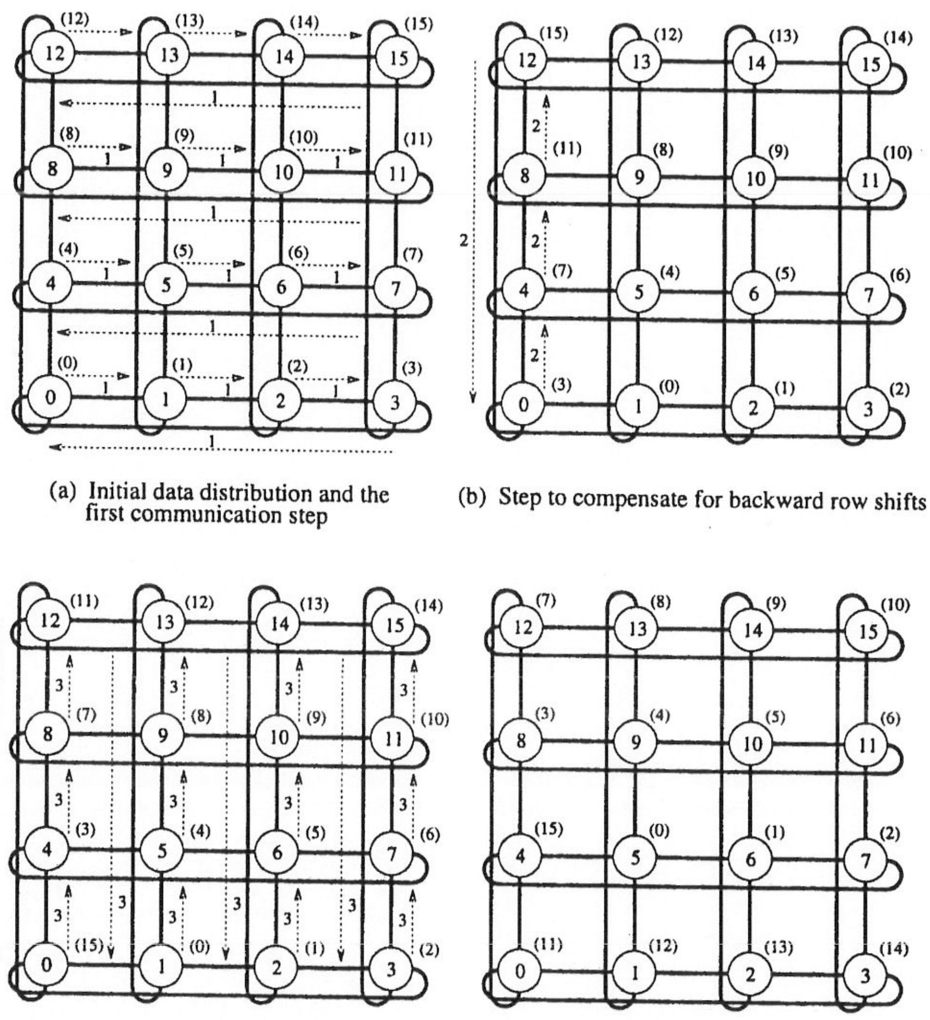
[redacted]

```
procedure ALL_TO_ALL_PERSONAL(d, my_id)
begin
    for i := 1 to 2^d − 1 do
    begin
        partner := my_id XOR i;
        send M_{my_id,partner} to partner;
        receive M_{partner,my_id} from partner;
    endfor;
end ALL_TO_ALL_PERSONAL
```

Program 3.8 A procedure to perform all-to-all personalized communication on a d-

CIRCULAR SHIFT (used in some matrix computations, string & image pattern matching)

- Ring: straightfwd
- MESH: stage ①: shift by (q mod $\sqrt{p}$) steps along rows
  - 1.5: data that wrapped around must shift 1 step along cols
  - stage ②: shift by $\lfloor q/\sqrt{p} \rfloor$ steps along cols.



(a) Initial data distribution and the first communication step

(b) Step to compensate for backward row shifts

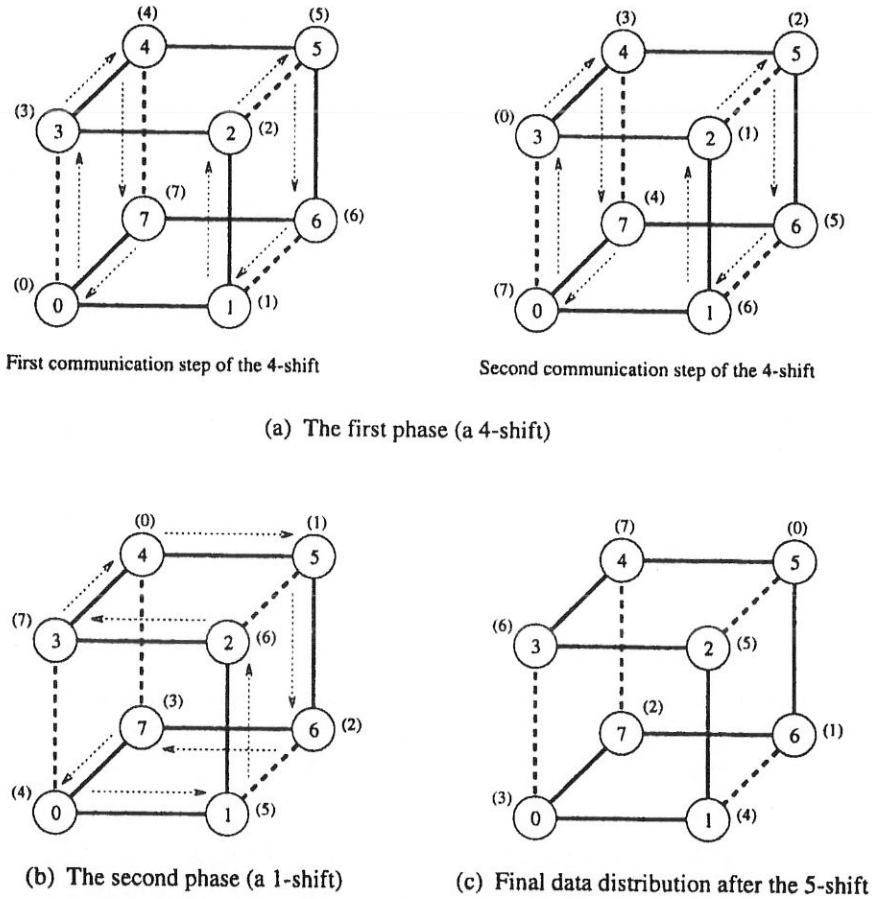
(c) Column shifts in the third communication step

(d) Final distribution of the data

Figure 3.22 The communication steps in a circular 5-shift on a 4 × 4 mesh.

$$T_{shift} = (t_s + t_w m)\left(2\left\lfloor \frac{\sqrt{p}}{2} \right\rfloor + 1\right)$$ [assuming both fwd & backward shifts]

- Map ring to HC using RGC
- Property: 2 processors at a distance of 2^i on ring are separated by exactly 2 links on HC (exception $i=0$)
- # communic phases in a q-shift = # 1's in the binary representation of q
- Total # steps in q-shift $= 2 \log p - 1$



First communication step of the 4-shift

Second communication step of the 4-shift

(a) The first phase (a 4-shift)

(b) The second phase (a 1-shift)

(c) Final data distribution after the 5-shift

Figure 3.23 The mapping of an eight-processor ring onto a three-dimensional hypercube to perform a circular 5-shift as a combination of a 4-shift and a 1-shift.

SF

- all communication in a step is congestion-free
[property of ring mapping:] procs whose distance on ring is power of 2 are in disjoint subrings on HC

$$T_{circ\text{-}shift} = (t_s + t_w m)(2 \log p - 1)$$

C1

- Use std labeling of processors, not RGC
- Use std E-cube routing to ensure congestion-free paths
- For q-shift, longest path has $(\log p - \gamma(q))$ links,

where $\gamma(q)$ = highest int j such that q is divisible by 2^j

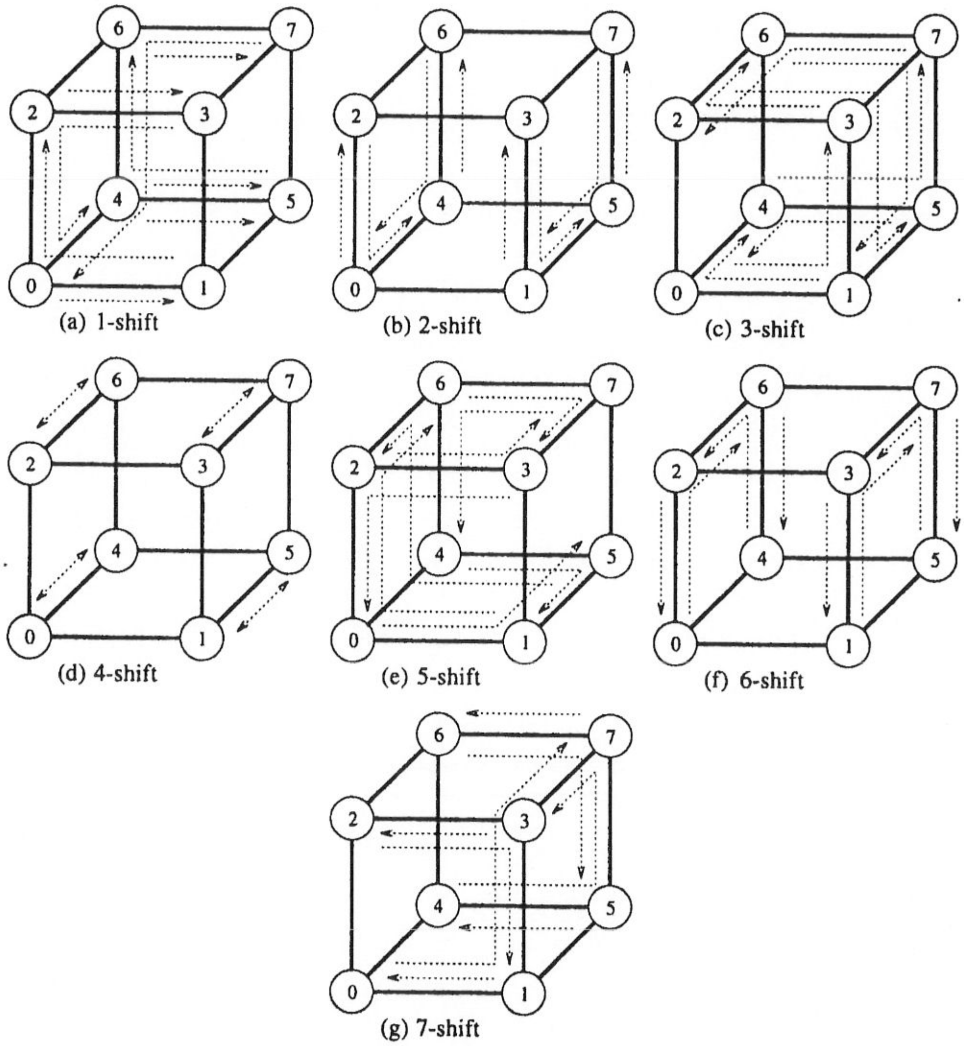


Figure 3.24 Circular q-shifts on an 8-processor hypercube for $1 \le q < 8$. CT

$$T_{circular\text{-}shift} = t_s + t_w m + t_h(\log p - \gamma(q))$$

- Routing Msg in Parts is faster?
- HC properties

1) log p ~~distinct~~ (disjoint) paths between any pair of procs.

If labels differ in l bits, l paths have l links each

$(\log p - l)$ paths have $(l+2)$ links each

Msg split into log p parts, each to dest along separate path, (longest 1st)

then dest can receive all data in max. $(2\log p)$ steps

$\Rightarrow 2(t_s \log p + t_w m)$ time

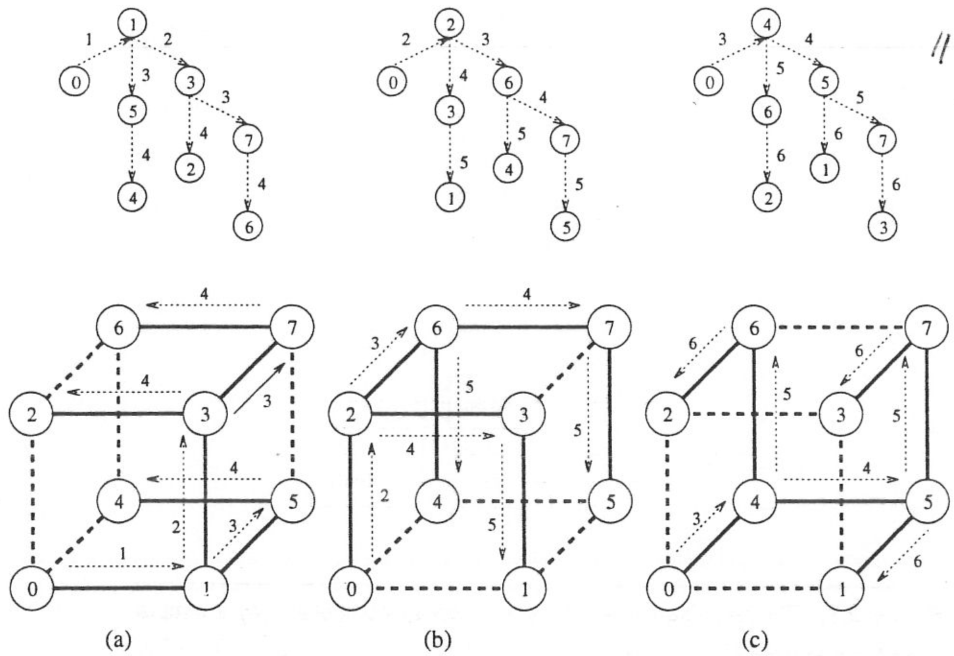
// scatter (Fig 3.16)

$t_s \log p + t_w \left(\frac{m}{p}\right)(p-1)$

// All-All BC

$t_s \log p + t_w \left(\frac{m}{p}\right)(p-1)$

Fig 3.12



1 → all BC

Figure 3.25 The six time-steps in one-to-all broadcast on an eight-processor hypercube with SF routing when the message is split into three parts that are routed separately on three different spanning binomial trees.

without splitting,

$T = (t_s + t_w m)\log p$

(Fig 3.5)

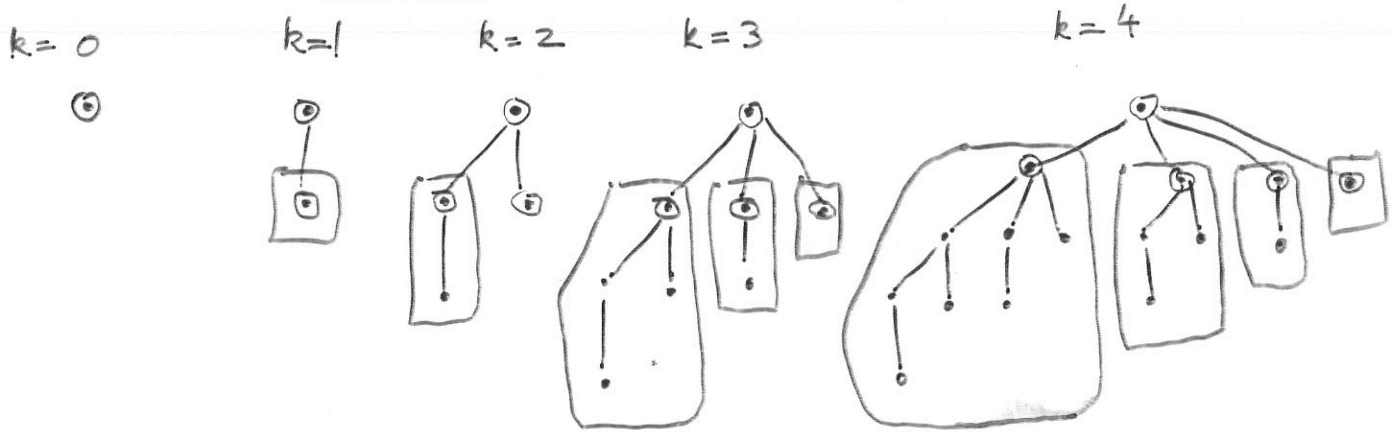
2) p-node binomial tree can be embedded into p-node HC w/ 1-1 node mapping

log p binomial trees rooted at 3 neighbors of source proc ϕ.

[Each processor (incl. root) sends out received msg to subtrees ~~decreasing~~ in the ~~sig~~ order of decreasing sizes of the subtrees

$\Rightarrow$ conflict-free msg passing

Binomial Tree, order k has 2^k nodes, height = k



children are roots of binomial trees of order k-1, k-2, -- 1, 0 tions 3.2–3.5 on different architectures with one-port communication and CT routing. The message size for each operation is m and the number of processors is p. The time for one-to-all broadcast on the hypercube is not optimal, and, as shown in Section 3.7.1 and Problem 3.24, can be improved to $2(t_s \log p + t_w m)$. In the hypercube expression for circular q-shift, $\gamma(q)$ is the highest integer j such that q is divisible by 2^j.

Operation	Ring	2-D Mesh (wraparound, square)	Hypercube
One-to-all broadcast	$(t_s + t_w m)\log p + t_h(p-1)$	$(t_s + t_w m)\log p + 2t_h(\sqrt{p}-1)$	$(t_s + t_w m)\log p$
All-to-all broadcast	$(t_s + t_w m)(p-1)$	$2t_s(\sqrt{p}-1) + t_w m(p-1)$	$t_s \log p + t_w m(p-1)$
One-to-all personalized	$(t_s + t_w m)(p-1)$	$2t_s(\sqrt{p}-1) + t_w m(p-1)$	$t_s \log p + t_w m(p-1)$
All-to-all personalized	$(t_s + t_w mp/2)(p-1)$	$(2t_s + t_w mp)(\sqrt{p}-1)$	$(t_s + t_w m)(p-1) + (t_h/2)p\log p$
Circular q-shift	$(t_s + t_w m)\lfloor p/2 \rfloor$	$(t_s + t_w m)(2\lfloor \sqrt{p}/2 \rfloor + 1)$	$t_s + t_w m + t_h(\log p - \gamma(q))$

For SF, above results are valid, except

1) 1 → all BC : ring $(t_s + t_w m)\lceil p/2 \rceil$

mesh $2(t_s + t_w m)\lceil \sqrt{p}/2 \rceil$

2) all → all personalized communication :

HC : $\left(t_s + t_w \frac{mp}{2}\right) \log p$